Microsoft

Microsoft Windows 2000 Server

Operating System

IP Telephony with TAPI 3.0

White Paper

Abstract

TAPI 3.0 is an evolutionary API providing convergence of both traditional PSTN telephony and IP telephony. IP telephony is an emerging set of technologies that enables voice, data, and video collaboration over existing LANs, WANs, and the Internet. TAPI 3.0 enables IP telephony on Microsoft® Windows® operating systems by providing simple and generic methods for making connections between two or more computers and accessing any media streams involved in the connection.

TAPI 3.0 supports standards-based H.323 conferencing and IP multicast conferencing. It uses the Windows 2000 operating system's Active Directory service to simplify deployment within an organization and includes quality-of-service (QoS) support to improve conference quality and network manageability.

C NTENTS

IP TELEPHONY

IP telephony is an emerging set of technologies that enables voice, data, and video collaboration over existing IP-based LANs, WANs, and the Internet.

Specifically, IP telephony uses open IETF and ITU standards to move multimedia traffic over any network that uses IP, offering users both flexibility in physical media (for example, POTS lines, ADSL, ISDN, leased lines, coaxial cable, satellite, and twisted pair) and flexibility of physical location. As a result, the same ubiquitous networks that carry Web, e-mail, and data traffic can be used to connect to individuals, businesses, schools, and governments worldwide.

TAPI 3.0 is an evolutionary API that supports convergence of both traditional PSTN telephony and telephony over IP networks.

Benefits of IP Telephony

IP telephony allows organizations and individuals to lower the costs of existing services, such as voice and broadcast video, while broadening their means of communication to include modem video conferencing, application sharing, and whiteboarding tools.

In the past, organizations have deployed separate networks to handle traditional voice, data, and video traffic. Each with different transport requirements, these networks were expensive to install, maintain, and reconfigure. Furthermore, since these networks were physically distinct, integration was difficult, if not impossible, limiting their potential usefulness.

IP telephony blends voice, video, and data by specifying a common transport, IP, for each, effectively collapsing three networks into one. The result is increased manageability, lower support costs, a new breed of collaboration tools, and increased productivity.

Possible applications for IP telephony include telecommuting, real-time document collaboration, distance learning, employee training, video conferencing, video mail, and video on demand.

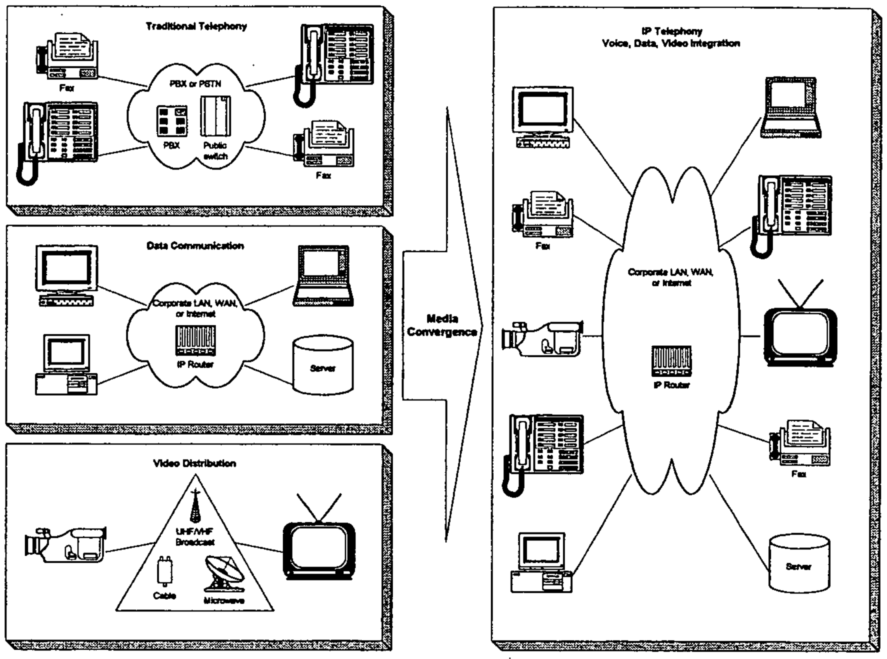


Figure 1. Media Convergence: voice, data, and video

INTRODUCTION T TAPI 3.0

As telephony and call control become more common in the desktop computer, a general telephony interface is needed to enable applications to access all the telephony options available on any computer. The media or data on a call must also be available to applications in a standard manner.

TAPI 3.0 provides simple and generic methods for making connections between two or more computers and accessing any media streams involved in that connection. It abstracts call-control functionality to allow different, and seemingly incompatible, communication protocols to expose a common interface to applications.

IP telephony is poised for explosive growth, as organizations begin a historic shift from expensive and inflexible circuit-switched public telephone networks to intelligent, flexible, and inexpensive IP networks. Microsoft, in anticipation of this trend, has created a robust computer telephony infrastructure, TAPI. Now in its third major version, TAPI is suitable for quick and easy development of IP telephony applications.

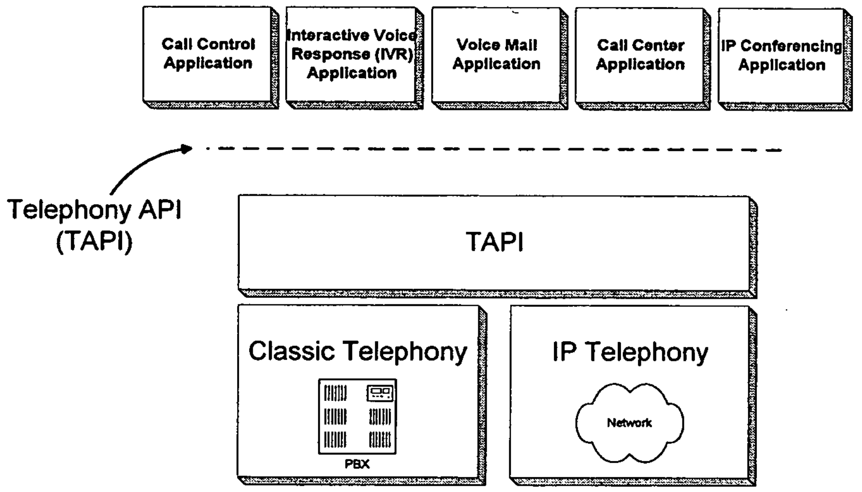


Figure 2. Convergence of IP and PSTN telephony

Inside TAPI 3.0

TAPI 3.0 integrates multimedia stream control with legacy telephony. Additionally, it is an evolution of the TAPI 2.1 API to the COM model, allowing TAPI applications to be written in any language, such as C/C++ or Microsoft® Visual Basic®.

Besides supporting classic telephony providers, TAPI 3.0 supports standard H.323 conferencing and IP multicast conferencing. TAPI 3.0 uses the Windows® 2000 Active Directory service to simplify deployment within an organization, and it supports quality-of-service (QoS) features to improve conference quality and network manageability.

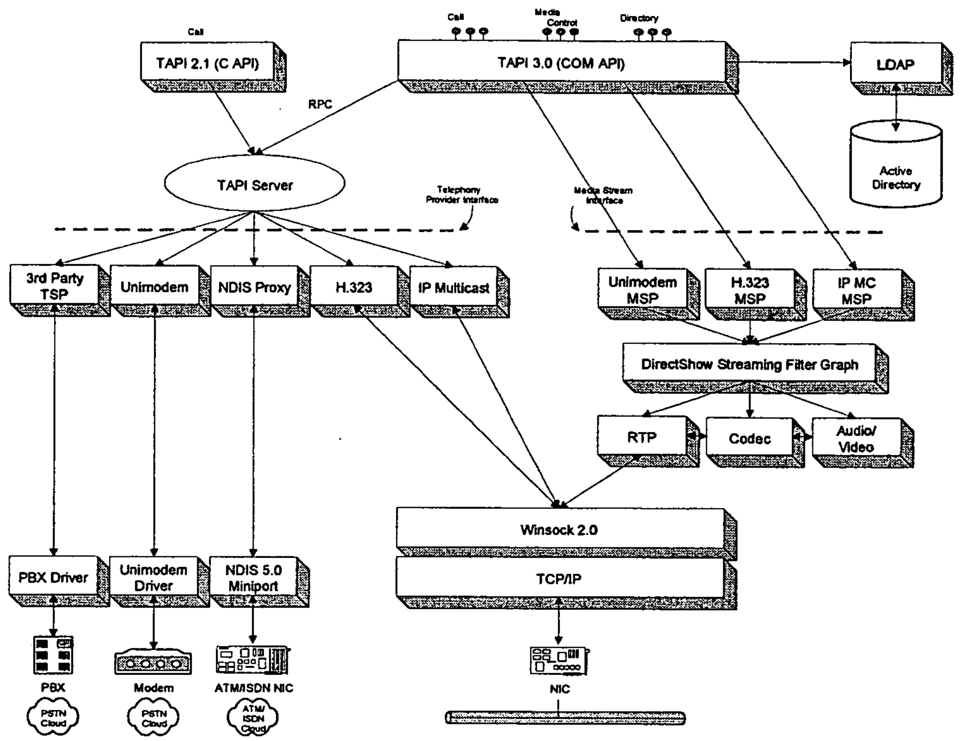


Figure 3. TAPI architecture

There are four major components to TAPI 3.0:

- TAPI 3.0 COM API
- TAPI Server
- Telephony Service Providers
- Media Stream Providers

In contrast to TAPI 2.1, the TAPI 3.0 API is implemented as a suite of COM objects. Moving TAPI to the COM model allows component upgrades of TAPI features. It also allows developers to write TAPI-enabled applications in any language.

The TAPI Server process (TAPISRV.EXE) abstracts the TSPI (TAPI Service Provider Interface) from TAPI 3.0 and TAPI 2.1, allowing TAPI 2.1 Telephony Service Providers to be used with TAPI 3.0, maintaining the internal state of TAPI.

Telephony Service Providers (TSPs) are responsible for resolving the protocol-independent call model of TAPI into protocol-specific call-control mechanisms. TAPI 3.0 provides backward compatibility with TAPI 2.1 TSPs. Two IP telephony service providers (and their associated MSPs) ship by default with TAPI 3.0: the H.323 TSP and the IP Multicast Conferencing TSP, which are discussed below.

TAPI 3.0 provides a uniform way to access the media streams in a call, supporting the DirectShow™ API as the primary media-stream handler. TAPI Media Stream Providers (MSPs) implement DirectShow interfaces for a particular TSP and are required for any telephony service that makes use of DirectShow streaming. Generic streams are handled by the application.

Call C ntrol M del

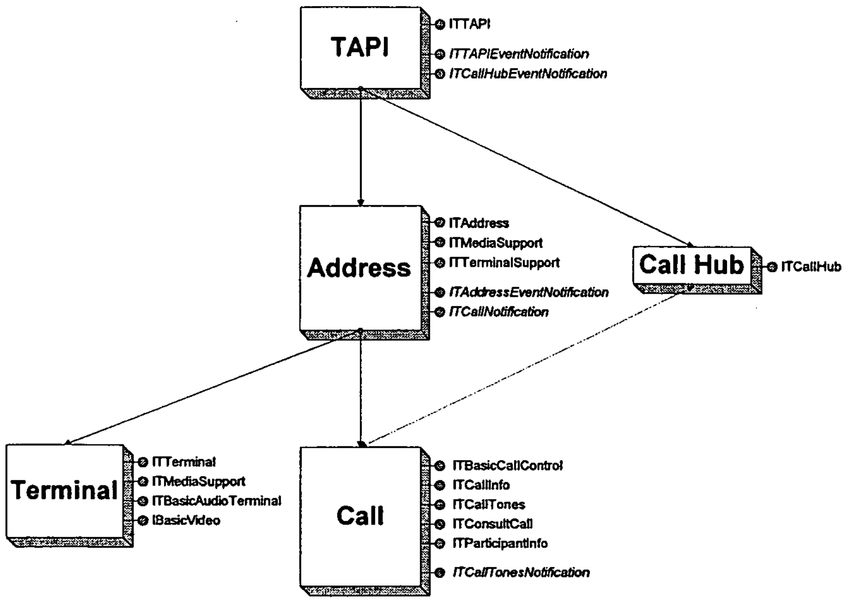


Figure 4. TAPI 3.0 object relationships

There are five objects in the TAPI 3.0 API:

- TAPI
- Address
- Terminal
- Call
- CallHub

The TAPI object is the application's entry point to TAPI 3.0. This object represents all telephony resources to which the local computer has access, allowing an application to enumerate all local and remote addresses.

An *Address* object represents the origination or destination point for a call. Address capabilities, such as media and terminal support, can be retrieved from this object. An application can wait for a call on an Address object or can create an outgoing call object from an Address object.

A *Terminal* object represents the sink, or renderer, at the termination or origination point of a connection. The Terminal object can map to hardware used for human interaction, such as a telephone or microphone, but can also be a file or any other device capable of receiving input or creating output.

The *Call* object represents an address's connection between the local address and one or more other addresses (This connection can be made directly or through a CallHub). The Call object can be imagined as a first-party view of a telephone call. All call control is done through the Call object. There is a call object for each member of a CallHub.

The *CallHub* object represents a set of related calls. A CallHub object cannot be created directly by an application—it is created indirectly when an incoming call is received through TAPI 3.0. Using a CallHub object, a user can enumerate the other participants in a call or conference, and possibly (because of the location independent nature of COM) perform call control on the remote Call objects associated with those users, subject to sufficient permissions:

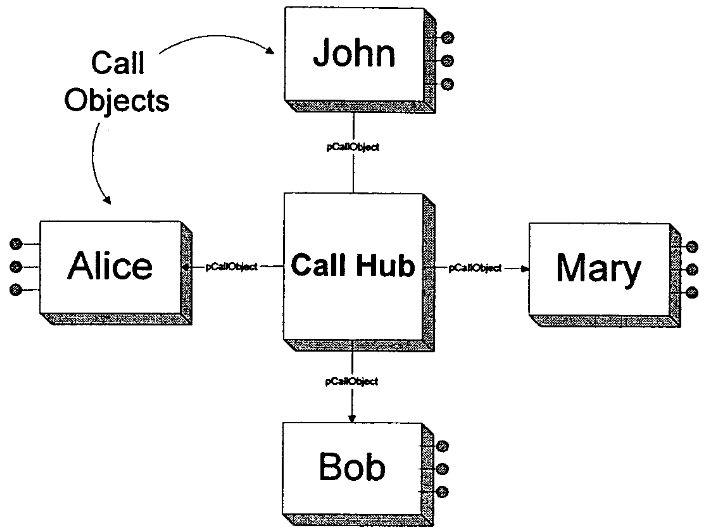


Figure 5. Call and CallHub object relationships

Using TAPI Objects

To place a call

1. Create and initialize a TAPI object.
2. Use the TAPI object to enumerate all available Address objects on a computer (for example, network cards, modems, and ISDN lines).
3. Enumerate the supported address types of each Address object (for example, a phone number, IP address, and so on).
4. Choose an Address object, based on queries for support for appropriate media (audio, video, and so on) and address types.
5. Use the **CreateCall** method of the Address object to create a Call object associated with a particular address.
6. Select appropriate Terminals on the Call object.
7. Call the **Connect** method of the Call object to place the call.

To answer a call

1. Create and initialize a TAPI object.
2. Use the TAPI object to enumerate all available Address objects on a computer (for example, network cards, modems, and ISDN lines).
3. Enumerate the supported address types of each Address object (for example, a phone number, IP address, etc.).
4. Choose an Address object, based on queries for support of appropriate media

(audio, video, and so on) and address types.

5. Register an interest in specific media types with the appropriate Address object.
6. Register a call event handler (that is, implement an **ITCallNotification** interface) with the Address object.
7. TAPI notifies the application of a new call through **ITCallNotification** and creates a Call object.
8. Select appropriate Terminals on the Call object.
9. Call the **Connect** method of the Call object to place the call.
10. Call the **Answer** method of the Call object to answer the call.

Media Streaming Model

The Windows® operating system provides an extensible framework for efficient control and manipulation of streaming media called DirectShow. DirectShow, through its exposed COM interfaces, provides TAPI 3.0 with unified stream control.

At the heart of DirectShow is a modular system of pluggable components called filters, arranged in a configuration called a filter graph. A component called the filter graph manager oversees the connection of these filters and controls the stream's data flow. Each filter's capabilities are described by a number of special COM interfaces called pins. Each pin instance can consume or produce streaming data, such as digital audio.

While COM objects are usually exposed in user-mode programs, the DirectShow streaming architecture includes an extension to the Windows driver model that allows the connection of media streams directly at the device-driver level. Figure 6 below shows a simple PSTN-to-IP bridge: A 64 Kbps voice stream from an ISDN line is compressed into a G.723 audio stream and passed to an RTP payload handler to be sent out over the network.

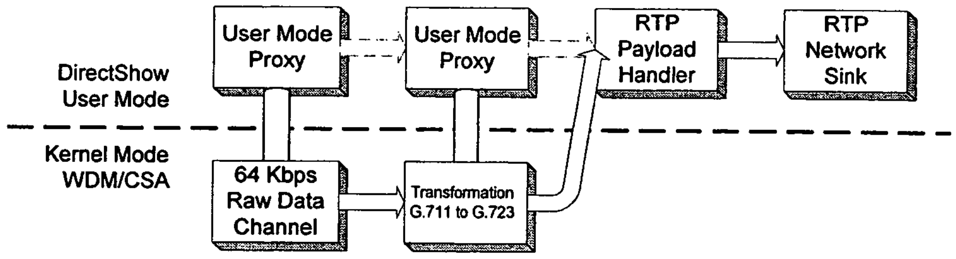


Figure 6. Sample DirectShow filter graph with user and kernel-mode components

These high-performance streaming extensions to the Windows Driver Model avoid user-to-kernel mode transitions and allow efficient routing of data streams between different hardware components at the device driver level. Each kernel mode filter is mirrored by a corresponding user-mode proxy that facilitates connection setup and can be used to control hardware-specific features.

DirectShow network filters extend the streaming architecture to computers connected on an IP network. The Real-time Transport Protocol (RTP), designed to carry real-time data over connectionless networks, transports TAPI media streams and provides appropriate time-stamp information. TAPI 3.0 includes a kernel-mode RTP network filter.

TAPI 3.0 utilizes this technology to present a unified access method for the media streams in multimedia calls. Applications can route these streams by manipulating corresponding filter graphs; they can also easily connect streams from multiple calls for bridging and conferencing capabilities.

H.323 COMMUNICATIONS IN TAPI 3.0

H.323 is a comprehensive International Telecommunications Union (ITU) standard for multimedia communications (voice, video, and data) over connectionless networks that do not provide a guaranteed quality of service, such as IP-based networks and the Internet. It provides for call control, multimedia management, and bandwidth management for point-to-point and multipoint conferences. H.323 mandates support for standard audio and video codecs and supports data sharing through the T.120 standard. Furthermore, the H.323 standard is network-, platform-, and application-independent, allowing any H.323-compliant terminal to interoperate with any other.

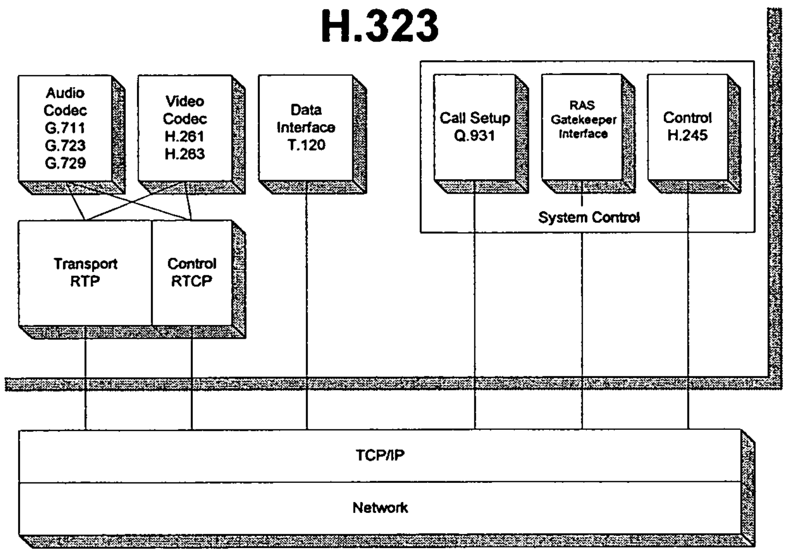


Figure 7. H.323 architecture

H.323 allows multimedia streaming over current packet-switched networks. To counter the effects of LAN latency, H.323 uses as a transport the Real-time Transport Protocol (RTP), an IETF standard designed to handle the requirements of streaming real-time audio and video over the Internet.

The H.323 standard specifies three command and control protocols:

- H.245 for call control
- Q.931 for call signaling
- The RAS (Registration, Admissions, and Status) signaling function

The H.245 control channel is responsible for control messages governing operation of the H.323 terminal, including capability exchanges, commands, and indications. Q.931 is used to set up a connection between two terminals, while RAS governs registration, admission, and bandwidth functions between endpoints and gatekeepers (RAS is not used if a gatekeeper is not present). See below for more information on gatekeepers.

H.323 defines four major components of an H.323-based communication system:

- Terminals

- Gateways
- Gatekeepers
- Multipoint Control Units (MCUs)

Terminals are the client endpoints on the network. All terminals must support voice communications; video and data support is optional.

A *Gateway* is an optional element in an H.323 conference. Gateways bridge H.323 conferences to other networks, communications protocols, and multimedia formats. Gateways are not required if connections to other networks or non-H.323-compliant terminals are not needed.

Gatekeepers perform two important functions that help maintain the robustness of the network: address translation and bandwidth management. Gatekeepers map LAN aliases to IP addresses and provide address lookups when needed. Gatekeepers also exercise call-control functions to limit the number of H.323 connections and the total bandwidth used by these connections, in an H.323 zone. A gatekeeper is not required in an H.323 system; however, if a gatekeeper is present, terminals must make use of its services.

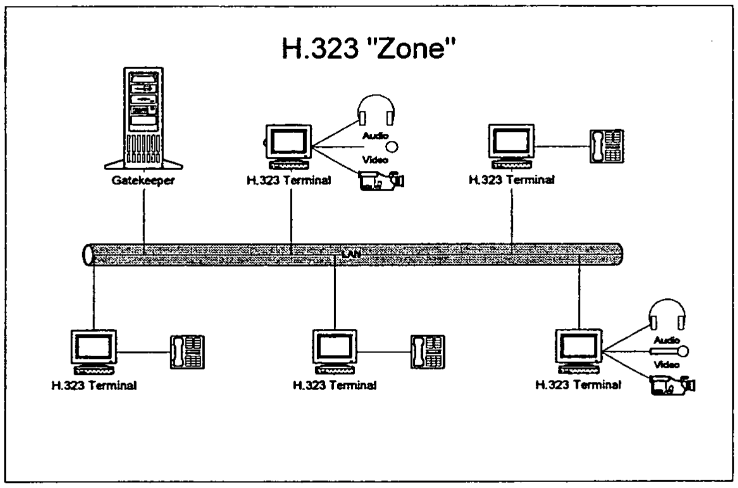


Figure 8. H.323 components

Multipoint Control Units (MCU) support conferences between three or more endpoints. An MCU consists of a required Multipoint Controller (MC) and 0 or more Multipoint Processors (MPs). The MC performs H.245 negotiations between all terminals to determine common audio and video processing capabilities, while the Multipoint Processor (MP) routes audio, video, and data streams between terminal endpoints.

Any H.323 client is guaranteed to support the following standards: H.261 and G.711. H.261 is an ITU-standard video codec designed to transmit compressed video at a rate of 64 Kbps and at a resolution of 176x44 pixels (QCIF). G.711 is an ITU-standard audio codec designed to transmit A-law and µ-law PCM audio at bit rates of 48, 56, and 64 Kbps.

Optionally, an H.323 client may support additional codecs: H.263 and G.723. H.263 is an ITU-standard video codec based on and compatible with H.261. It offers improved compression over H.261 and transmits video at a resolution of 176 x 44 pixels (QCIF). G.723 is an ITU-standard audio codec designed to operate at very low bit rates.

TAPI 3.0 H.323 Telephony Service Provider

The H.323 Telephony Service Provider (with its associated Media Stream Provider) allows TAPI-enabled applications to engage in multimedia sessions with any H.323-compliant terminal on the local area network.

Specifically, the H.323 Telephony Service Provider (TSP) implements the H.323 signaling stack. The TSP accepts a number of different address formats, including name, computer name, and e-mail address.

The H.323 MSP is responsible for constructing the DirectShow filter graph for an H.323 connection (including the RTP, RTP payload handler, codec, sink, and renderer filters).

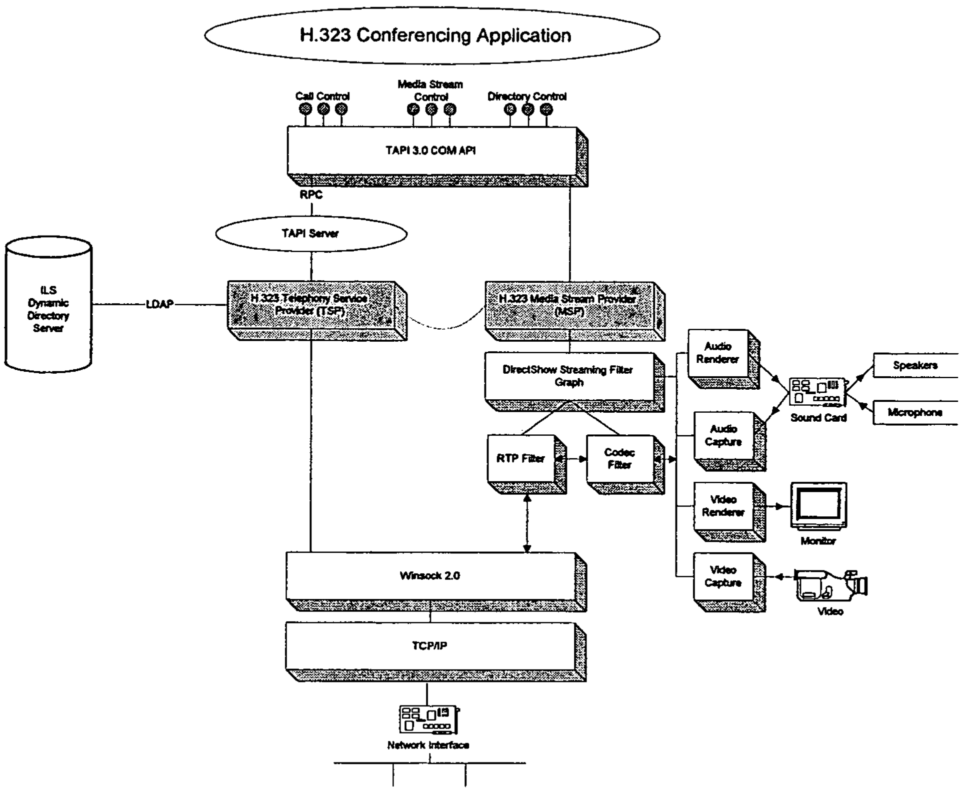


Figure 9. H.323 TSP architecture

Integration with the Windows 2000 Active Directory

H.323 telephony is complicated by the fact that a user's network address (in this case, a user's IP address) is highly volatile and cannot be counted on to remain unchanged between H.323 sessions. The TAPI H.323 TSP uses the services of the Windows 2000 Active Directory to perform user-to-IP address resolution. Specifically, user-to-IP mapping information is stored and continually refreshed using the Internet Locator Service (ILS) Dynamic Directory, a real-time server component of the Active Directory.

The following user scenario illustrates IP address resolution in the H.323 TSP:

1. John wishes to initiate an H.323 conference with Alice, another user on the LAN. Once Alice's video conferencing application creates an Address object and puts it in listen mode, Alice's IP address is added to the Windows 2000 Active Directory by the H.323 TSP. This information has a finite time-to-live (TTL) and is refreshed at regular intervals through the Lightweight Directory Access Protocol (LDAP):

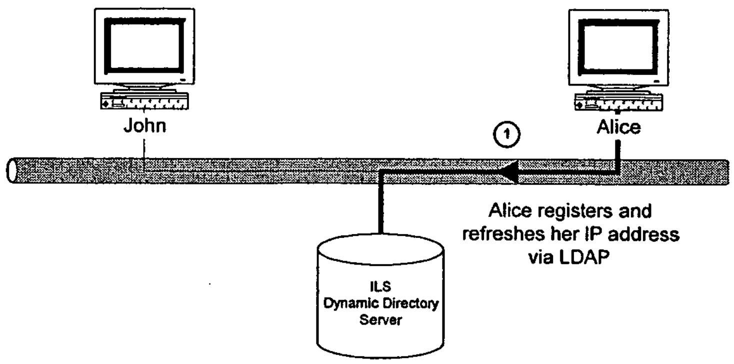

John
Alice
1
Alice registers and refreshes her IP address via LDAP
ILS Dynamic Directory Server

Figure 10. Alice registers and refreshes her IP address

2. John's H.323 TSP then queries the ILS Dynamic Directory server for Alice's IP address. Specifically, John queries for any and all RTPerson objects in the directory associated with Alice:

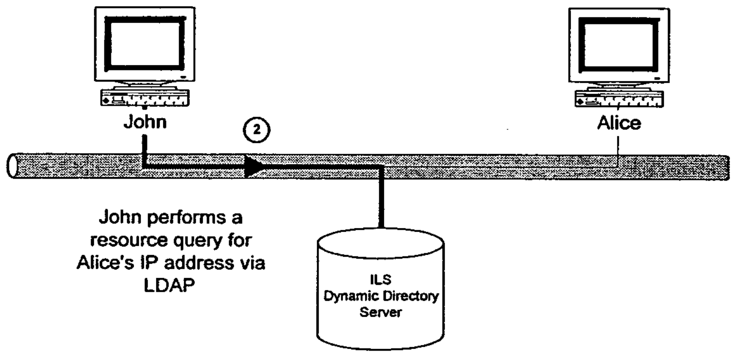

John
Alice
2
John performs a resource query for Alice's IP address via LDAP
ILS Dynamic Directory Server

Figure 11. John queries for Alice's IP address

3. With Alice's up-to-date IP address, John initiates an H.323 call to Alice's computer, and H.323-standard negotiations and media selection occurs between the peer TSPs on both computers. Once capability negotiations have been completed, both H.323 Media Stream Providers (MSPs) construct appropriate DirectShow filter graphs, and all media streams are passed off to DirectShow to handle. The conference then begins.

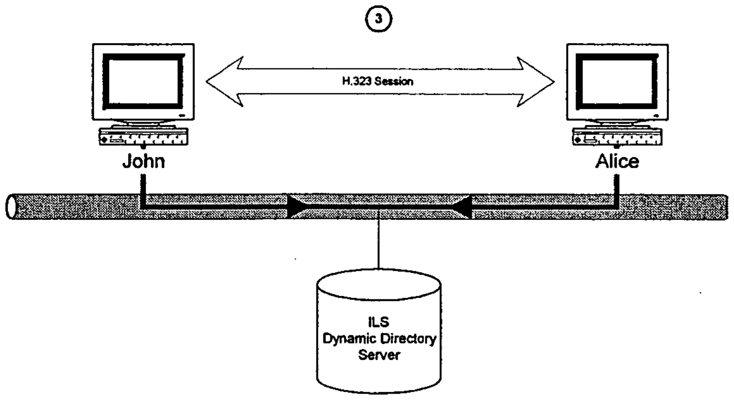


Figure 12. Once capability negotiations have been completed, the conference begins

IP MULTICAST C NFERENCING IN TAPI 3.0

IP multicast is an extension of IP that allows for efficient group communication. IP multicast arose out of the need for a lightweight, scalable conferencing solution that solved the problems associated with real-time traffic over a datagram, best-effort network. There are many advantages to using IP multicast: scalability, fault tolerance, robustness, and ease of setup.

The IP multicast conferencing model incorporates the following key features:

- No global coordination is needed to add and remove members from a conference.
- To reach a multicast group, a user sends data to a single multicast IP address. No knowledge of the other users in a group is necessary.
- To receive data, users register their interest in a particular multicast IP address with a multicast-aware router. No knowledge of the other users in a group is necessary.
- Routers hide the multicast implementation details from the user.
- Traditional connection-oriented conferencing suffers from a number of problems:
    - *User complexity*: Users must know the location of every user they wish to converse with, limiting scalability and fault-tolerance and rendering it difficult for users to add and remove themselves from a conference.
    - *Wasted bandwidth*: A user wishing to broadcast data to n users must send data through n connections, as shown in the following illustration:

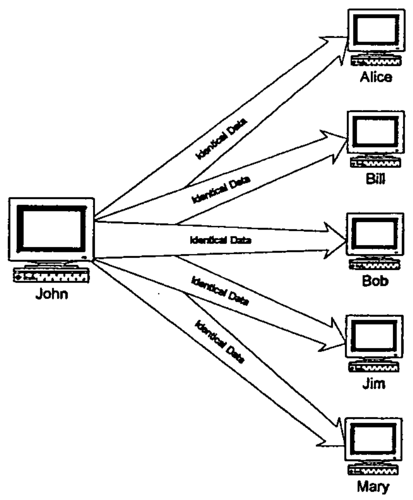


Figure 13. Network topology: sender's view

The total bandwidth required for multiparty conferences in which all users are sending data goes up as the square of the number of parties involved, leading to huge scalability problems. IP Multicast takes advantage of the actual network topology to eliminate the transmission of redundant data down the same communications links.

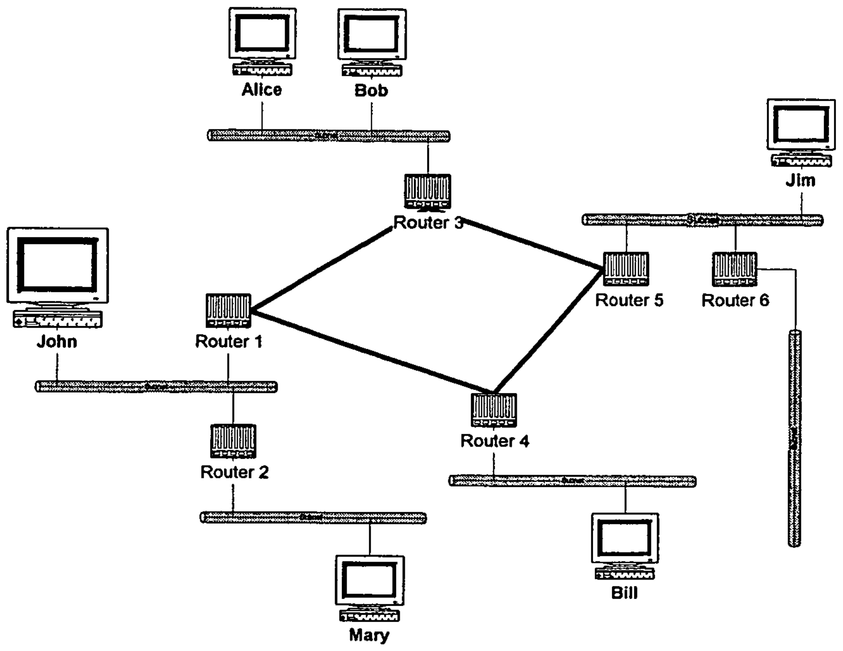


Figure 14. Actual network topology

IP multicast implements a lightweight, session-based communications model, which places relatively little burden on conference users. Using IP multicast, users send only one copy of their information to a group IP address that reaches all recipients. IP multicast is designed to scale well as the number of participants expands—adding one more user does not add a corresponding amount of bandwidth. Multicasting also results in a greatly reduced load on the sending server.

IP multicast routes these one-to-many data streams efficiently by constructing a spanning tree, in which there is only one path from one router to any other. Copies of the stream are made only when paths diverge:

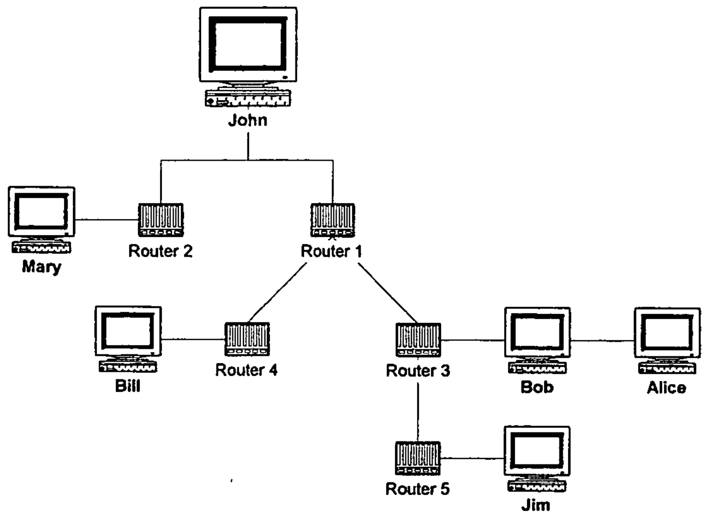


Figure 15. IP multicast using a spanning tree

Without multicasting, the same information must either be carried over the network multiple times, one time for each recipient, or broadcast to everyone on the network, consuming unnecessary bandwidth and processing.

IP multicast uses Class-D Internet Protocol addresses to specify multicast host groups, ranging from 224.0.0.0 to 239.255.255.255. Both permanent and temporary group addresses are supported. Permanent addresses are assigned by the Internet Assigned Numbers Authority (IANA) and include 224.0.0.1, the all-hosts group used to address all multicast hosts on the local network, and 224.0.0.2, which addresses all routers on a LAN. The range of addresses between 224.0.0.0 and 224.0.0.255 is reserved for routing and other low-level network protocols. Other addresses and ranges have been reserved for applications, such as 224.0.13.000 to 224.0.13.255 for Net News (for more information, see RFC 1700, "Assigned Numbers" at ftp://ftp.internic.net/rfc/rfc1700.txt).

The transport protocol for IP Multicast is RTP (Real-time Transport Protocol), which provides a standard multimedia header giving time stamp, sequence numbering, and payload format information.

Applications for IP multicast include video and audio conferencing, telecommuting, database and Web-site replication, distance learning, dissemination of stock quotes, and collaborative computing. At present, the largest demonstration of the capabilities of IP multicast is the Internet MBONE (Multicast Backbone).

The MBONE is an experimental, global multicast network layered on top of the physical Internet. It has been in existence for about five years, and presently carries IETF meetings, NASA space shuttle launches, music, concerts, and many other live meetings and performances (for more information, see http://www.mbone.com).

TAPI 3.0 IP Multicast Confer ncing TSP

The IP Multicast Conferencing TSP is chiefly responsible for resolving conference names to IP multicast addresses, using the Session Description Protocol (SDP) conference descriptors stored in the ILS Dynamic Directory Conference Server. It is complemented by the Rendezvous conference controls, described below.

The IP Multicast Conferencing MSP is responsible for constructing an appropriate DirectShow filter graph for an IP multicast connection (including RTP, RTP payload handler, codec, sink, and renderer filters).

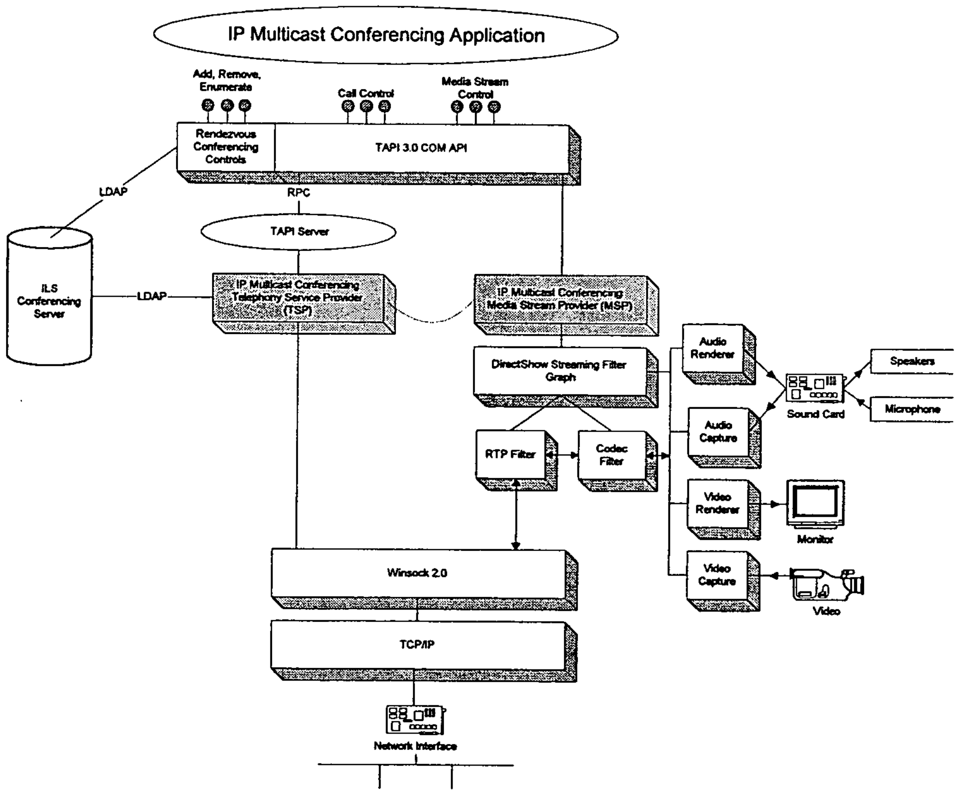


Figure 16. IP multicast conferencing architecture

Integration with the Windows 2000 Active Directory

TAPI 3.0 uses the IETF-standard Session Description Protocol to advertise IP multicast conferences across the enterprise. SDP descriptors are stored in the Windows 2000 Active Directory—specifically, in the ILS Dynamic Directory Conference Server. SDP is discussed in more detail below. In contrast to the Dynamic Directory servers utilized by the H.323 TSP, there is only one ILS Conference Server per enterprise, since conference announcements are not continually refreshed, therefore consuming little bandwidth.

TAPI 3.0's IP multicast conference mechanism is illustrated in the following scenario, in which John wishes to initiate a multicast conference:

1. John's TAPI 3.0-enabled application uses the Rendezvous Controls (discussed in more detail below) to create an SDP session descriptor on the ILS Conference Server. The SDP descriptor contains, among other things, the conference name, start and end times, the IP multicast address of the conference, and the media types used for the conference.

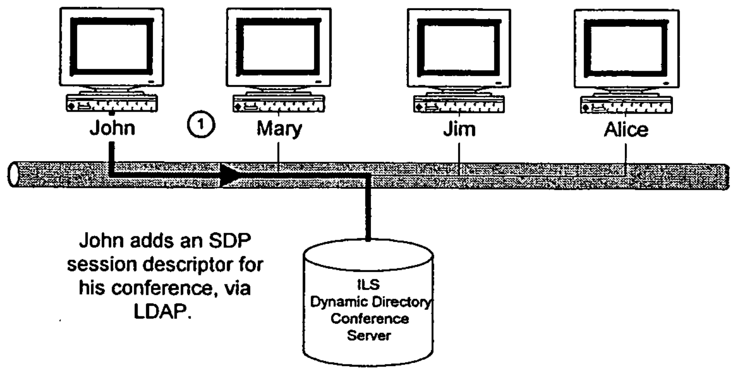

John
1
Mary
Jim
Alice
John adds an SDP session descriptor for his conference, via LDAP.
ILS Dynamic Directory Conference Server

Figure 17. John adds an SDP session descriptor

2. Jim queries the ILS Conference Server for SDP descriptors of conferences matching his criteria:

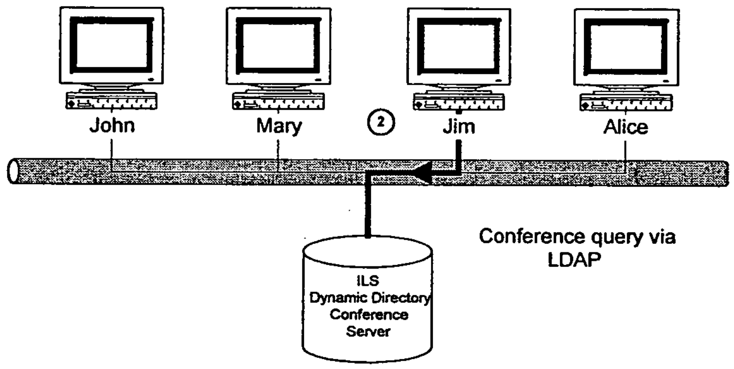

John
Mary
2
Jim
Alice
Conference query via LDAP
ILS Dynamic Directory Conference Server

Figure 18. Jim queries the ILS Conference Server

3. Mary and Alice perform similar queries and use the SDP information they receive to decide to participate in John's conference. With the multicast IP address of the conference, they join the multicast host group:

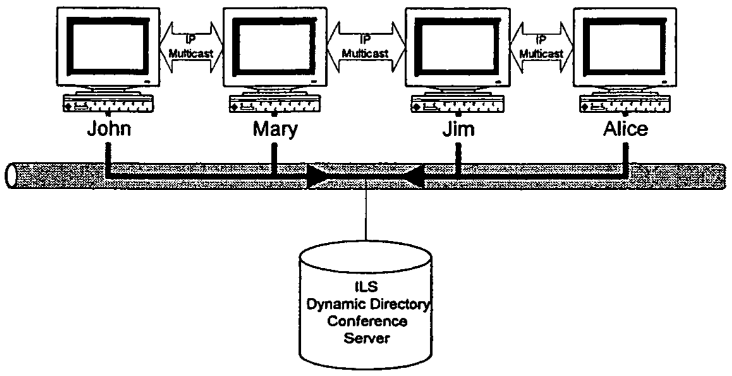


Figure 19. Mary and Alice join the conference

TAPI 3.0 Rendezvous Controls

The Rendezvous Controls are a set of COM components that abstract the concept of a conference directory, providing a mechanism to advertise new multicast conferences and to discover existing ones. They provide a common schema (SDP) for conference announcement, as well as scriptable interfaces, authentication, encryption, and access-control features.

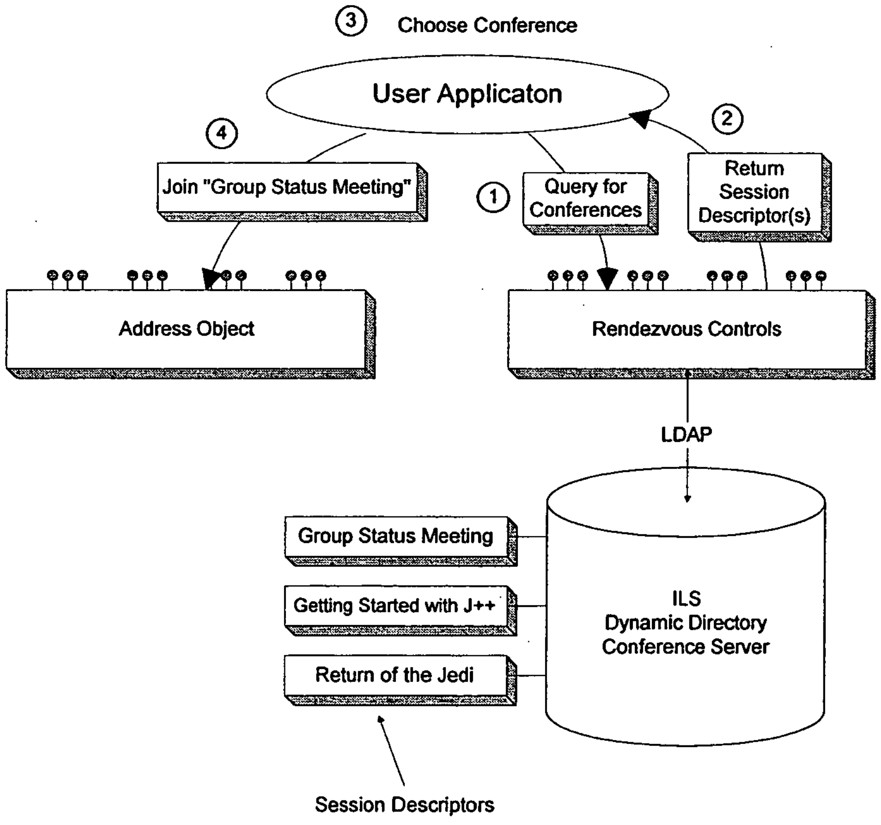


Figure 20. Joining a conference, using the Rendezvous controls

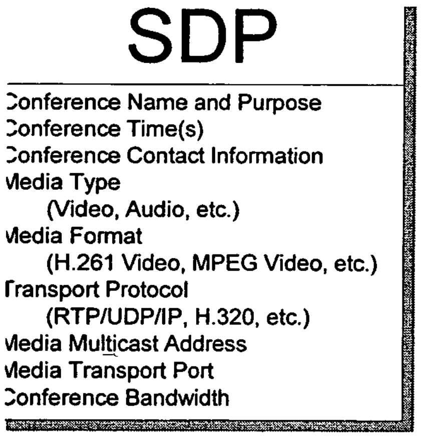
The user may add, delete, and enumerate multicast conferences stored on an ILS Conference Server through the Rendezvous controls. These controls manipulate conference data through the Lightweight Directory Access Protocol (LDAP).

Joining a conference is illustrated in Figure 20 above. The conferencing application uses the Rendezvous controls to obtain session descriptors for the conferences that match the user's criteria (1,2). Access control lists (ACLs) protect each of the stored conference announcements, and whether or not an announcement is visible and accessible depends upon the user's credentials.

Once the user has chosen a conference (3), the user application searches for all Address objects that support the address type Multicast Conference Name. The application then uses the conference name from the SDP descriptor as a parameter to the **CreateCall** method of the appropriate Address object (4), passes the appropriate Terminal objects to the returned Call object, and calls Call->**Connect**.

The Rendezvous controls store the conference information on an ILS Conference Server in a format defined by the Session Description Protocol (SDP), an IETF standard for announcing multimedia conferences. The purpose of SDP is to publicize sufficient information about a conference (time, media, and location information) to allow prospective users to participate if they choose. Originally designed to operate over the Internet MBONE (IP Multicast Backbone), SDP has been integrated by TAPI 3.0 with the Windows 2000 Active Directory, thus extending its functionality to local area networks.

An SDP descriptor advertises the following information about a conference:


SDP

Conference Name and Purpose
Conference Time(s)
Conference Contact Information
Media Type
  (Video, Audio, etc.)
Media Format
  (H.261 Video, MPEG Video, etc.)
Transport Protocol
  (RTP/UDP/IP, H.320, etc.)
Media Multicast Address
Media Transport Port
Conference Bandwidth

Figure 21. General SDP attributes

A session description is broken into three main parts: a single Session Description, 0 or more Time Descriptions, and 0 or more Media Descriptions. The Session Description contains global attributes that apply to the whole conference or all media streams. Time Descriptions contain conference start, stop, and repeat-time information, and Media Descriptions contain details that are specific to a particular media stream.

While traditional IP multicast conferences operating over the MBONE have advertised conferences using a push model based on the Session Announcement Protocol (SAP), TAPI 3.0 employs a pull-based approach, using Windows 2000 Active Directory services. This approach offers numerous advantages, among them bandwidth conservation and ease of administration. See the Integration with Windows 2000 Active Directory section for details.

Conference Security Model

TAPI 3.0's conference security system addresses the following needs:

- Controlling who can create, delete, and view conference announcements.
- Preventing conference eavesdropping.

TAPI 3.0 uses the security features of the Windows 2000 Active Directory and LDAP to provide for secure conferencing over insecure networks, such as the Internet. Each object in the Active Directory can be associated with an Access Control List (ACL) specifying object-access rights on a user or group basis. By associating ACLs with SDP conference descriptors, conference creators can specify who can enumerate and view conference announcements. User authentication is provided by the Windows 2000 security subsystem.

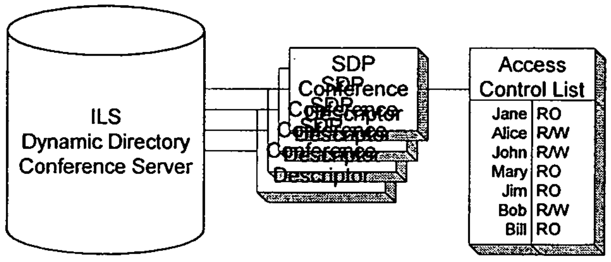


Figure 22. SDPs and ACLs

Session Descriptors are transmitted from the ILS Conference Server to the user over LDAP in encrypted form, through a Secure Sockets Layer (SSL) connection, ensuring that the SDP is safe from eavesdroppers:

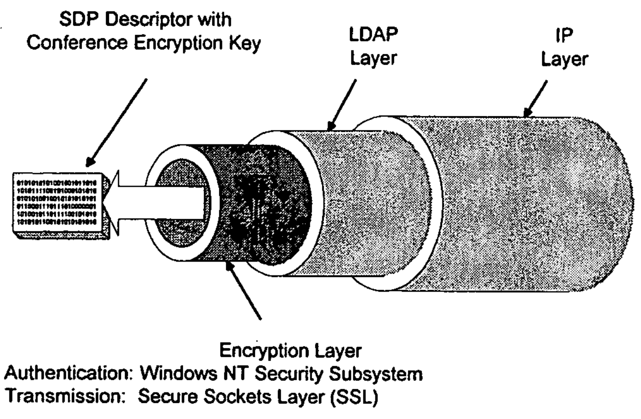

SDP Descriptor with
Conference Encryption Key
LDAP
Layer
IP
Layer
Encryption Layer
Authentication: Windows NT Security Subsystem
Transmission: Secure Sockets Layer (SSL)

Figure 23. Distribution of the SDP

IP multicast makes no provision for authenticating users; any user may anonymously join a multicast host group. To keep conferences private, TAPI 3.0 allows an IP multicast conference to be encrypted, with the encryption key distributed from within the conference descriptor. Only users with sufficient permissions have access to a conference's SDP descriptor and, therefore, the multicast encryption key. Once an authenticated user fetches the encryption key, he or she can participate in the conference.

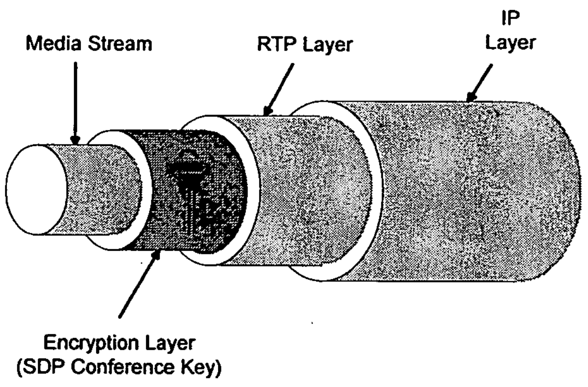


Figure 24. Encrypted multicast stream

QUALITY OF SERVICE

In contrast to traditional data traffic, multimedia streams, such as those used in IP telephony or videoconferencing, may be extremely bandwidth- and delay-sensitive, imposing unique quality-of-service (QoS) demands on the underlying networks that carry them. Unfortunately, IP, with a connectionless, best-effort delivery model, does not guarantee delivery of packets in order, in a timely manner, or at all. In order to deploy real-time applications over IP networks with an acceptable level of quality, certain bandwidth, latency, and jitter requirements must be guaranteed and met so that multimedia traffic can coexist with traditional data traffic on the sam network.

Bandwidth: Multimedia data, and in particular video, requires orders of magnitude more bandwidth than traditional networks can handle. An uncompressed NTSC video stream, for example, can require upwards of 220 megabits per second. Even compressed, a handful of multimedia streams can completely overwhelm any other traffic on the network.

Latency: The amount of time that a multimedia packet takes to get from the source to the destination (latency) has a major impact on the perceived quality of the call. There are many contributors towards latency, including transmission delays, queuing delays in network equipment, and delays in host protocol stacks. Latency must be minimized in order to maintain a certain level of interactivity and to avoid unnatural pauses in conversation.

Jitter: In contrast to data traffic, real-time multimedia packets must arrive in order and on time to be of any use to the receiver. Variations in packet arrival time (jitter) must be below a certain threshold to avoid dropped packets (and therefore irritating shrieks and gaps in the call). Jitter, by determining receive buffer sizes, also affects latency.

Coexistence: In comparison with multimedia traffic, data traffic is relatively bursty, and arrives in unpredictable chunks (for instance, when someone opens a Web page, or downloads a file from an FTP site). Aggregations of such bursts can clog routers and cause gaps in multimedia conferences, leaving calls at the mercy of everyone on the network (including other IP telephony users). Multimedia bandwidth must be protected from data traffic, and vice versa.

Public-switched telephone networks guarantee a minimum quality of service by allocating static circuits for every telephone call. Such an approach is simple to implement, but wastes bandwidth, lacks robustness, and makes voice, video, and data integration difficult. Furthermore, circuit-switched data paths are impossible to create using a connectionless network such as IP.

QoS support on IP networks offers the following benefits:

- Support for real-time multimedia applications.
- Assurance of timely transfers of large amounts of data.
- The ability to share the network in a manner that avoids starving applications of bandwidth.

Quality of rvic and TAPI 3.0

Quality of service in TAPI 3.0 is handled through the DirectShow RTP filter, which negotiates bandwidth capabilities with the network, based on the requirements of the DirectShow codecs associated with a particular media stream. These requirements are indicated to the RTP filter by the codecs through its own QoS interface. The RTP filter then uses the COM Winsock2 GQoS interfaces to indicate, in an abstract form, its QoS requirements to the Winsock2 QoS service provider (QoS SP). The QoS SP, in turn, invokes various QoS mechanisms appropriate for the application, the underlying media, and the network, in order to guarantee appropriate end-to-end QoS. These mechanisms include:

- The Resource Reservation Protocol (RSVP)
- Local Traffic Control:
  - Packet Scheduling
  - 802.1p
  - Appropriate Layer-2 signaling mechanisms
- IP Type of Service and DTR header settings

RSVP

The Resource Reservation Protocol (RSVP) is an IETF standard designed to support resource (for example, bandwidth) reservations through networks of varying topologies and media. Through RSVP, a user's QoS requests are propagated to all routers along the data path, allowing the network to reconfigure itself (at all network levels) to meet the desired level of service.

The RSVP protocol engages network resources by establishing *flows* throughout the network. A *flow* is a network path associated with one or more senders, one or more receivers, and a certain QoS. A sending host wishing to send data that requires a certain QoS broadcasts, through an RSVP-enabled Winsock Service Provider, path messages toward the intended recipients. These path messages, which describe the bandwidth requirements and relevant parameters of the data to be sent, are propagated to all intermediate routers along the path.

A receiving host, interested in this particular data, confirms the flow (and the network path) by sending reserve messages through the network, describing the bandwidth characteristics of data it should receive from the sender. As these reserve messages propagate back toward the sender, intermediate routers, based on bandwidth capacity, decide whether or not to accept the proposed reservation and commit resources. If an affirmative decision is made, the resources are committed and reserve messages are propagated to the next hop on the path from source to destination.

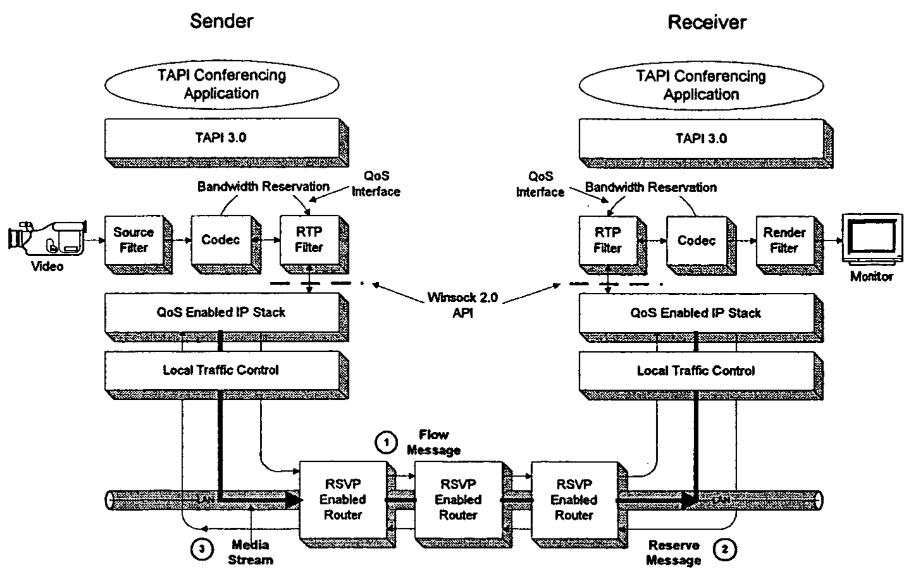


Figure 25. Resource reservation with TAPI 3.0

Local Traffic Control

Packet Scheduling: This mechanism can be used in conjunction with RSVP (if the underlying network is RSVP-enabled) or without RSVP. Traffic is identified as belonging to one flow or another, and packets from each flow are scheduled in accordance with the traffic-control parameters for the flow. These parameters generally include a scheduled rate (token bucket parameter) and some indication of priority. The former is used to pace the transmission of packets to the network. The latter is used to determine the order in which packets should be submitted to the network when congestion occurs.

801.2p: Traffic control can also be used to determine the 802.1 *User Priority* value (a MAC header field used to indicate relative packet priority) to be associated with each transmitted packet. 802.1p-enabled switches can then give preferential treatment to certain packets over others, providing additional QoS support at the data-link-layer level.

Layer 2 Signaling Mechanisms: In response to Winsock 2 QoS APIs, the QoS service provider may invoke additional traffic-control mechanisms, depending on the specific underlying data-link layer. It may signal an underlying ATM network, for instance, to set up an appropriate virtual circuit for each flow. When the underlying medium is a traditional 802 shared media network, the QoS service provider may extend the standard RSVP mechanism to signal a Subnet Bandwidth Manager (SBM). The SBM provides centralized bandwidth management on shared networks.

IP Typ of S rvice

Each IP packet contains a three-bit *Precedence* field, which indicates the priority of the packet. An additional field can be used to indicate a delay, throughput, or reliability preference to the network. Local traffic control can be used to set these bits in the IP headers of packets on particular flows. As a result, packets belonging to a flow are treated appropriately later by three devices on the network. These fields are analogous to 802.1p priority settings but are interpreted by higher-layer network devices.

ENTERPRISE DEPLOYMENT OF TAPI 3.0 IP TELEPHONY INFRASTRUCTURE

TAPI 3.0 has been designed to scale from the smallest business up to the largest organizations, while taking advantage of the Windows 2000 Active Directory to bring IP telephony to the enterprise.

Enterprise Layout for TAPI 3.0 IP Telephony

Figure 26 below illustrates the enterprise layout for a sample enterprise with two sites connected through the Internet. The ILS Dynamic Directory Servers and the ILS Dynamic Directory Conference Server, as explained above, provide functionality for point-to-point and multiparty conferencing. IP telephony clients can use video and audio capture equipment and can also support legacy telephones through the use of a PSTN add-in card.

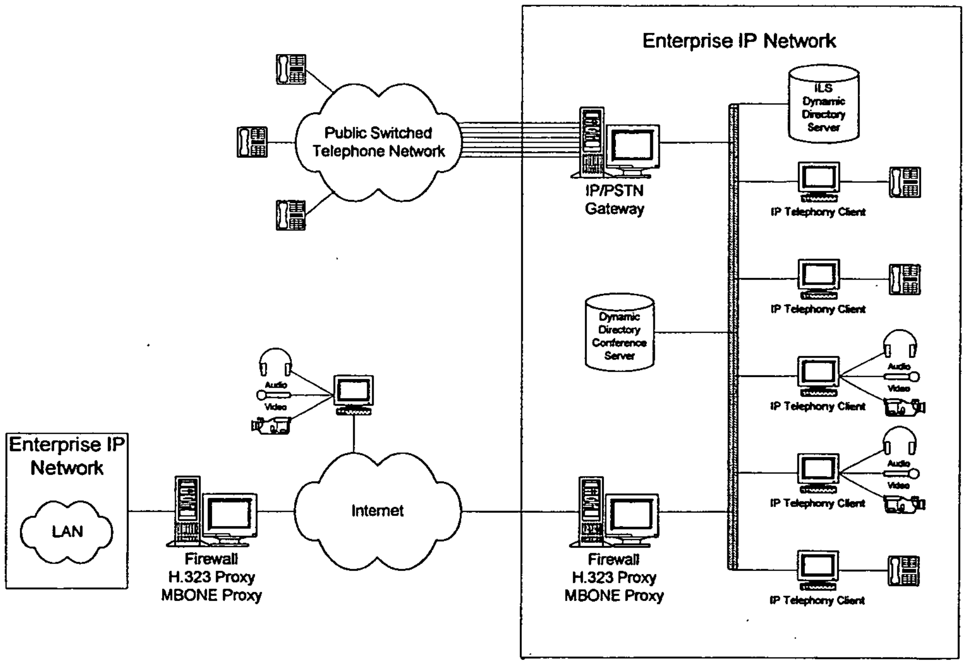

Enterprise IP Network
ILS Dynamic Directory Server
Public Switched Telephone Network
IP/PSTN Gateway
IP Telephony Client
Dynamic Directory Conference Server
Audio
Video
Enterprise IP Network
LAN
Internet
Firewall H.323 Proxy MBONE Proxy
Firewall H.323 Proxy MBONE Proxy

Figure 26. Enterprise layout for IP telephony

The IP/PSTN gateway digitizes incoming analog voice calls from PSTN lines and encapsulates them in H.323 streams, and vice versa, providing users with the ability to send and receive legacy voice calls through existing telephony infrastructure.

The H.323 Proxy allows H.323 clients to have connectivity with the Internet by forwarding H.323 streams through the enterprise firewall. This enables H.323 Internet, intranet, and business-to-business connectivity.

The function of the IP Multicast Proxy is somewhat similar to that of the H.323 Proxy—to forward multicast conference packets, but it also furnishes clients with the ability to propagate selected conference announcements to and from the Internet.

The IP Multicast Proxy monitors conference announcements stored on the ILS Dynamic Directory Conference Server and broadcasts conferences with appropriate scope and security attributes to the Internet, using the Session Announcement Protocol (SAP).

Conversely, the IP Multicast Proxy listens for appropriate conferences from those broadcast over the Internet and populates the ILS Dynamic Directory Conference Server with these announcements. In this manner, the IP Multicast Proxy allows users conference connectivity over the Internet while ensuring the confidentiality and security of private conferences.

Windows 2000 Active Directory Layout for IP Telephony

As discussed earlier, the H.323 TSP uses the services of the ILS Dynamic Directory component of the Active Directory to remove the burden of name-to-IP translation from the user.

At the network level, the Windows 2000 Active Directory model treats an organization as a collection of sites. Sites are regions of good connectivity, such as subnets or LANs, and typically correlate with physical locales, such as campuses.

For bandwidth and performance reasons, ILS servers are typically distributed across the enterprise, one per site, with each ILS server (or a replicating cluster of severs) being responsible for maintaining user-to-IP mappings for their site. To conserve bandwidth, these volatile mappings are not replicated across sites.

TAPI 3.0 uses the Active Directory to associate users with particular ILS servers. Users wishing to place an IP telephone call first consult the Global Catalog (a replicated subset of the Active Directory) for the User object of the person they wish to call. The Telephony container in the User object contains the name of the ILS server for that user's site, which is then queried for the IP address in question.

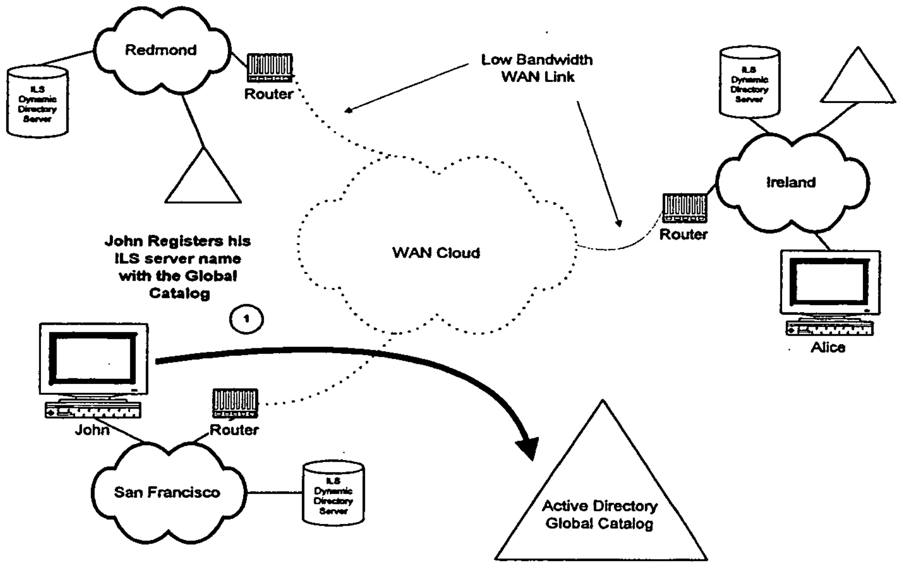

Redmond
ILS Dynamic Directory Server
Router
Low Bandwidth WAN Link
ILS Dynamic Directory Server
Ireland
Router
WAN Cloud
John Registers his ILS server name with the Global Catalog
1
Alice
John
Router
San Francisco
ILS Dynamic Directory Server
Active Directory Global Catalog

Figure 27. IP telephone call process

The following scenario illustrates enterprise deployment of the TAPI 3.0 directory infrastructure. In this example, Alice wants to initiate an H.323 call to John.

1. John previously registered his ILS server name with the Active Directory Global Catalog. Upon initialization, John's H.323 TSP queries the Global Catalog for the Subnet object associated with his computer and derives what site John's subnet and computer belong to. The TSP then fetches the name of the ILS server (or cluster of replicating servers) from DNS records and stores this information in John's User object.
2. Subsequently, Alice's H.323 TSP queries her local copy of the Global Catalog for the name of John's ILS server:

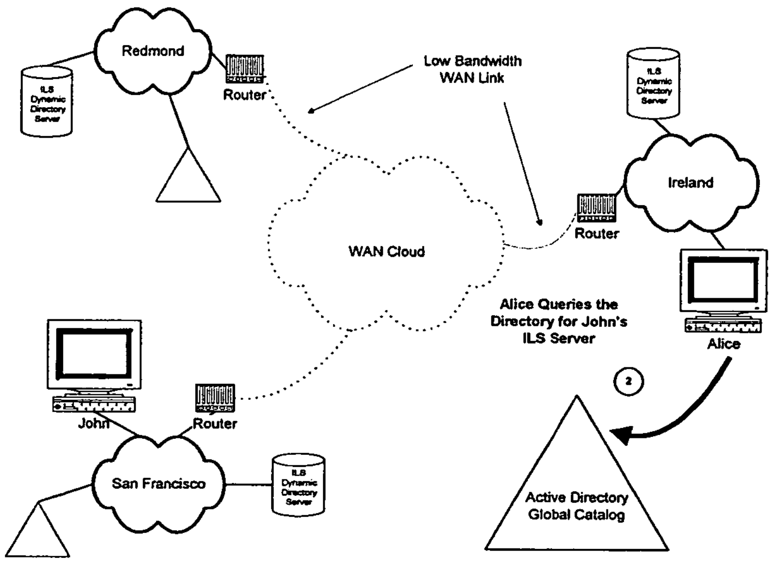

Redmond
ILS Dynamic Directory Server
Router
Low Bandwidth WAN Link
ILS Dynamic Directory Server
Ireland
Router
WAN Cloud
Alice Queries the Directory for John's ILS Server
Alice
John
Router
San Francisco
ILS Dynamic Directory Server
2
Active Directory Global Catalog

Figure 28. Alice's TSP queries her local copy of the Global catalog.

3. Alice's H.323 TSP then queries John's ILS server across the WAN for John's current IP address:

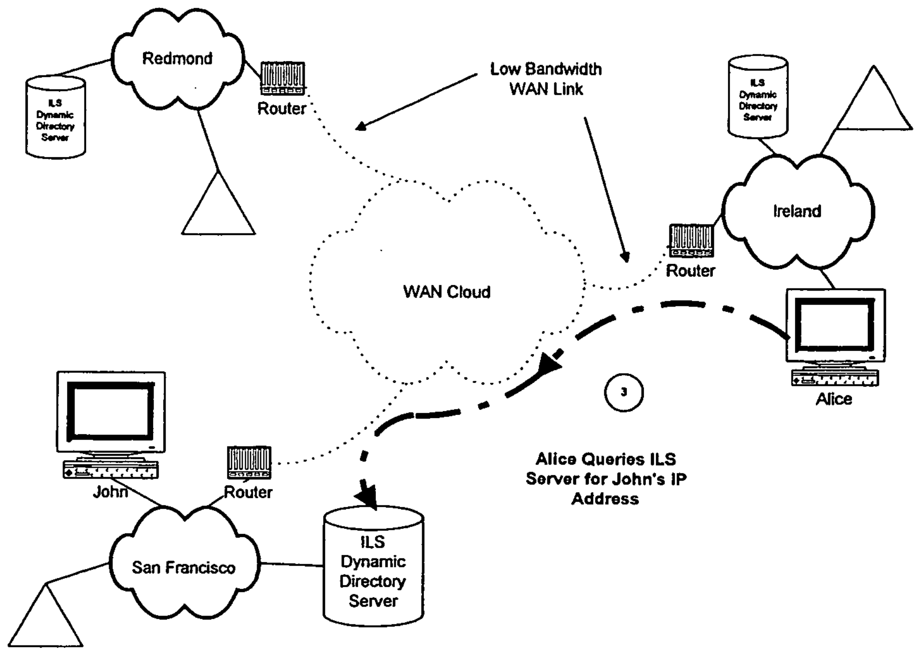

Redmond
ILS Dynamic Directory Server
Router
Low Bandwidth WAN Link
ILS Dynamic Directory Server
Ireland
Router
WAN Cloud
Alice
3
Alice Queries ILS Server for John's IP Address
John
Router
San Francisco
ILS Dynamic Directory Server

Figure 29. Alice's TSP queries for John's IP address.

4. Alice then initiates an H.323 session with John:

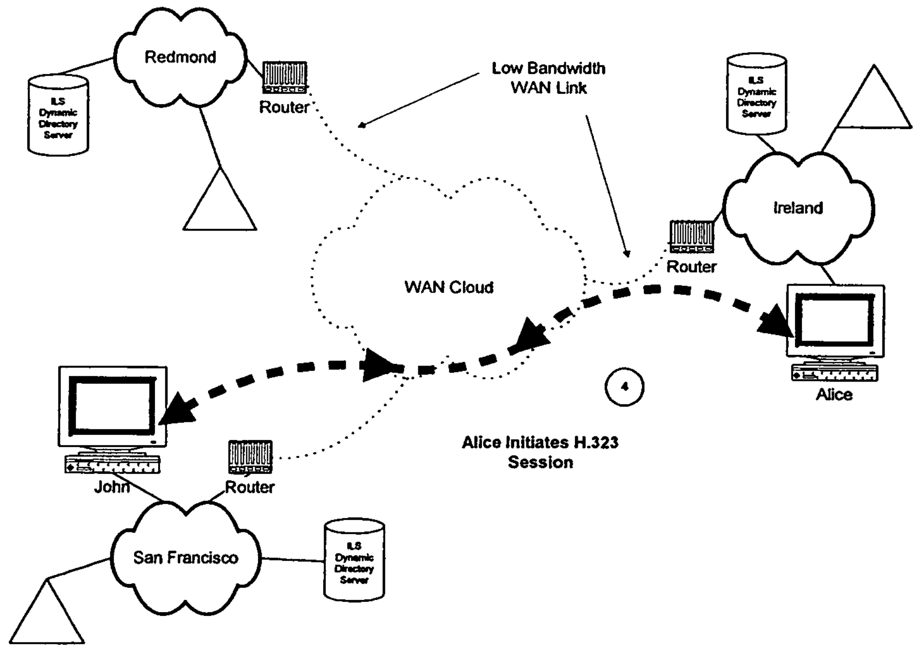

Redmond
ILS Dynamic Directory Server
Router
Low Bandwidth WAN Link
ILS Dynamic Directory Server
Ireland
Router
WAN Cloud
Alice
4
Alice Initiates H.323 Session
John
Router
San Francisco
ILS Dynamic Directory Server

Figure 30. Alice initiates a session with John.

The call abstraction inherent in TAPI allows this ILS and Active Directory interaction to occur transparently both to the user and to the TAPI 3.0-enabled application.

TAPI 3.0 AND NETMEETING 2.0

Microsoft NetMeeting is a conferencing and collaboration tool designed for the Internet or intranet. NetMeeting also provides a set of programming interfaces for adding conferencing functionality to your applications. It helps small and large organizations take full advantage of the global reach of the Internet or corporate intranet for real-time communications and collaboration by combining IP telephony and conferencing functionality. Connecting to other NetMeeting users is made easy with the Microsoft Internet Locator Server (ILS), enabling participants to call each other from a dynamic directory within NetMeeting or from a Web page. While connected on the Internet or corporate intranet, participants can communicate with both voice and video, work together on virtually any Windows-based application, exchange or mark up graphics on an electronic whiteboard, transfer files, or use the text-based chat program. For more information on Microsoft NetMeeting 2.0, see http://www.microsoft.com/netmeeting.

Microsoft NetMeeting 2.0 has the following features:

H.323 standards–based voice and video conferencing. Real-time, point-to-point audio conferencing over the Internet or corporate intranet enables a user to make voice calls to associates and organizations around the world. NetMeeting voice conferencing offers many features, including half-duplex and full-duplex audio support for real-time conversations, automatic microphone sensitivity level setting to ensure that meeting participants hear each other clearly, and microphone muting, which lets users control the audio signal sent during a call. This voice conferencing supports network TCP/IP connections.

Support for the H.323 protocol enables interoperability between NetMeeting 2.0 and other H.323-compatible voice clients. The H.323 protocol supports the ITU G.711 and G.723 audio standards and IETF RTP and RTCP specifications for controlling audio flow to improve voice quality. On MMX-enabled computers, NetMeeting uses the MMX-enabled voice codecs to improve performance for voice compression and decompression algorithms. This results in lower CPU use and improved voice quality during a call.

With NetMeeting 2.0, a user can send and receive real-time visual images with another conference participant, using any video for Windows-compatible equipment. They can share ideas and information face –to face and use the camera to instantly view items, such as hardware or devices, that the user chooses to display in front of the lens. Combined with the video and data capabilities of NetMeeting 2.0, a user can both see and hear the other conference participant, as well as share information and applications. This H.323 standards–based video technology is also compliant with the H.261 and H.263 video codecs.

Multipoint data conferencing using T.120. Two or more users can communicate and collaborate as a group in real time. Participants can share applications, exchange information through a shared clipboard, transfer files, collaborate on a shared whiteboard, and use a text-based chat feature. Support for the T.120 data conferencing standard also enables interoperability with other T.120-based products and services. The following features comprise multipoint data conferencing:

Application sharing: A user can share a program running on one computer with other participants in the conference. Participants can review the same data or information and see the actions as the person sharing the application works on the program (for example, editing content or scrolling through information.) Participants can share Windows-based applications transparently without any special knowledge of the application capabilities.

The person sharing the application can choose to collaborate with other conference participants, and they can take turns editing or controlling the application. Only the person sharing the program needs to have the given application installed on their computer.

Shared Clipboard: The shared clipboard enables a user to exchange its contents with other participants in a conference, using familiar cut, copy, and paste operations. For example, a participant can copy information from a local document and paste the contents into a shared application as part of a group collaboration.

File Transfer: With the file transfer capability, a user can send a file in the background to one or all of the conference participants. When one user drags a file into the main window, the file is automatically sent to each person in the conference; they can then accept or decline receipt. This file transfer capability is fully compliant with the T.127 standard.

Whiteboard: Multiple users can simultaneously collaborate using the whiteboard to review, create, and update graphic information. The whiteboard is object-oriented (versus pixel-oriented), enabling participants to manipulate the contents by clicking and dragging with the mouse. In addition, they can use a remote pointer or highlighting tool to point out specific contents or sections of shared pages.

Chat: A user can type text messages to share common ideas or topics with other conference participants or record meeting notes and action items as part of a collaborative process. Participants in a conference can also use chat to communicate in the absence of audio support. A whisper feature lets a user have a separate, private conversation with another person during a group chat session.

NetMeeting 2.0 Software Development Kit. This SDK enables developers to integrate this conferencing functionality directly into their applications or Web pages. This open development environment supports international communication and conferencing standards and enables interoperability with products and services from multiple vendors.

Also in the NetMeeting SDK are APIs to add nonstandard codecs and to access ILS servers through LDAP, as well as an ActiveX™ control to simplify adding conferencing capabilities to Web pages.

For more information on the *Microsoft NetMeeting 2.0 Software Development Kit*, see http://www.microsoft.com/netmeeting/sdk.

F atures of TAPI 3.0 and N tM ting 2.0

TAPI 3.0 and NetMeeting 2.0 both support core IP telephony capabilities. Each platform offers unique benefits: TAPI 3.0 seamlessly integrates traditional telephony with IP telephony, providing a COM-based, protocol-independent call-control and data streaming infrastructure. NetMeeting 2.0 SDK supports T.120 conferencing and application sharing in addition to IP Telephony. Applications using TAPI 3.0 and the NetMeeting 2.0 API interoperate using H.323 audio and video conferencing.

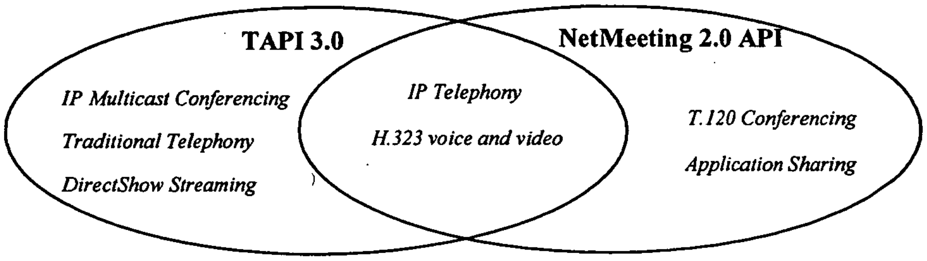


Figure 31. TAPI 3.0 and NetMeeting 2.0 interoperability

Using TAPI 3.0 and NetMeeting 2.0

Because TAPI 3.0 and NetMeeting 2.0 both support core IP telephony capabilities (including support for H.323), developers may want to consider the following guidelines when choosing an API for their IP telephony applications:

TAPI 3.0. This is the API to use if you are doing IP telephony in your application. TAPI 3.0 is especially valuable in the world of client/server computer telephony integration, for combining IP telephony with traditional telephony, and for IP multicast of voice and video.

NetMeeting 2.0 API. This is the API to use if you are doing real-time collaboration and want to integrate voice, video, and data conferencing into your application. The NetMeeting API is useful for applications that want to integrate application sharing, whiteboard functionality, and multipoint file transfer with voice and video sessions.

Microsoft

Microsoft Windows 2000 Server

Operating System

TAPI 3.0 Connection and Media Services

White Paper

Abstract

TAPI 3.0 is an evolutionary API providing convergence of both traditional PSTN telephony and IP telephony. IP telephony is an emerging set of technologies that enables voice, data, and video collaboration over existing LANs, WANs, and the Internet. TAPI 3.0 enables IP telephony on the Microsoft® Windows® operating system by providing simple and generic methods for making connections between two or more computers and accessing any media streams involved in the connection.

To provide connection and media services, TAPI uses Telephony Service Providers (TSPs) and Media Stream Providers (MSPs), collectively called Service Providers (SPs). This paper, intended for developers, reviews SPs as they exist in TAPI 2.x, and discusses the enhancements provided in TAPI 3.0.

INTRODUCTION

IP telephony is an emerging set of technologies that enables voice, data, and video collaboration over existing IP-based LANs, WANs, and the Internet.

Specifically, IP telephony uses open IETF and ITU standards to move multimedia traffic over any network that uses IP, offering users both flexibility in physical media (for example, POTS lines, ADSL, ISDN, leased lines, coaxial cable, satellite, and twisted pair) and flexibility of physical location. As a result, the same ubiquitous networks that carry Web, e-mail, and data traffic can be used to connect to individuals, businesses, schools, and governments worldwide.

The Microsoft® Windows® Telephony Application Programming Interface (TAPI 3.0) is an evolutionary API that supports convergence of both traditional PSTN telephony and telephony over IP networks. Intended for developers, this paper describes service providers (SPs) as they exist in TAPI 2.*x* and the enhancements that are in TAPI 3.0. A service provider can either be a Telephony Service Provider (TSP) or a Media Service Provider (MSP). TSPs are responsible for resolving the protocol-independent call model of TAPI into protocol-specific call-control mechanisms. MSPs implement Microsoft DirectShow® interfaces for a particular TSP and are required for any telephony service that makes use of DirectShow streaming.

This paper assumes a familiarity with TAPI 2.*x* and the Component Object Model (COM).

THE TAPI 3.0 ARCHITECTURE

The Microsoft Windows 2000 TAPI 3.0 architecture is shown in Figure 1.

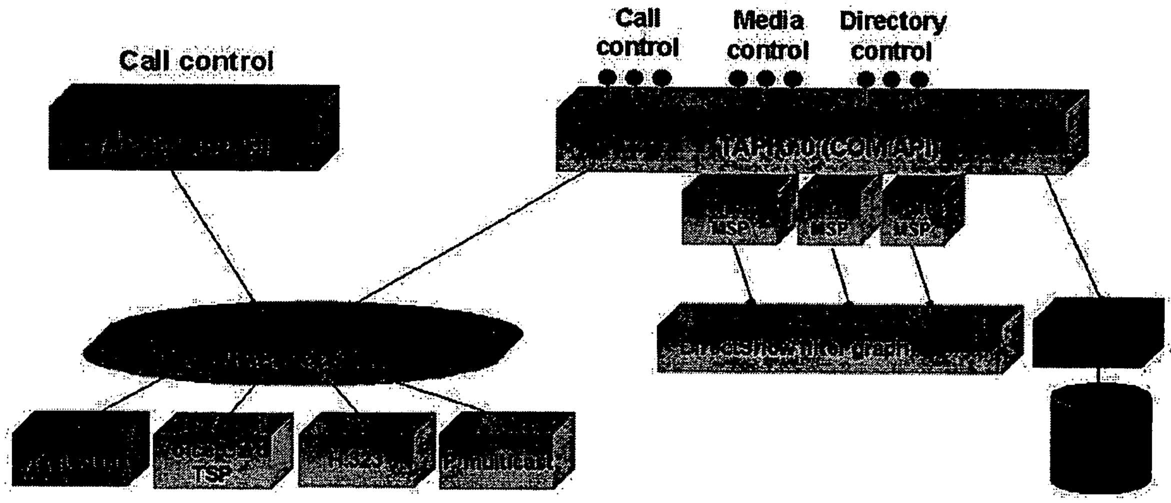


Figure 1. Windows 2000 TAPI 3.0 architecture

The left side of the drawing shows the components shipping today in TAPI 2.*x*. The right side shows the new components that are part of TAPI 3.0. The TAPI 3.0 COM API allows object-oriented, language-neutral software development. It is built on top of the TAPI server. The C API (to the left) provides call functionality. The new TAPI 3.0 COM API provides not only call functionality but also media access, directory access, and terminal access as well. (Terminals are the devices on a computer that capture and render audio and video. A sound card and a video camera are examples of devices that can be terminals.)

At the application level, there are COM interfaces for call control, media control, and directory control. There are also COM interfaces at the SP level to support services such as IP telephony and media control (the latter is exposed as part of th MSP). Therefore, in TAPI 3.0, COM objects are a part of the SP. Some of these service provider objects and interfaces will be aggregated through TAPI objects while the MSPs will directly provide others to the applications.

The architecture diagram demonstrates that TAPI 3.0 incorporates the features of TAPI 2.*x*, and enhances them. This means that existing TSPs will work with TAPI 3.0 and that developers can build on knowledge they already have to either expand existing SPs or build new ones.

A REVIEW F TAPI 2.*X*

As a prelude to discussing TAPI 3.0 SPs, this paper first reviews the basics of TAPI 2.*x* SPs, pointing out differences and parallels with TAPI 3.0 as they occur, beginning with a review of the TAPI 2.x distributed architecture.

TAPI 2.*x* Remote Service Provision

Figure 2 illustrates the distributed architecture that was first introduced in TAPI 2.1.

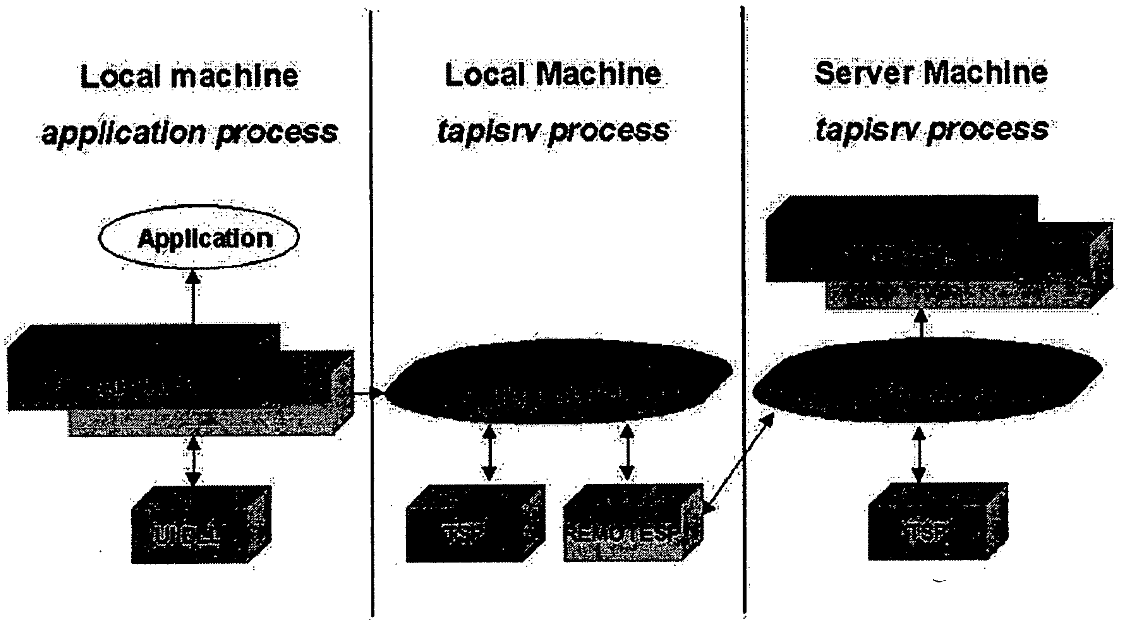


Figure 2. Distributed architecture introduced in TAPI 2.1

Service providers operate in a distributed environment. This distribution is given transparently by *remotesp*. Transparent distribution means the SP neither knows nor cares if it's operating on the same computer as the client application. Just as this had significance in TAPI 2.*x*, for the user interface (UI) component, so does it still have significance in TAPI 3.0, when providing media services.

The UI DLL in TAPI 2.*x* uses exactly the same model as the MSP in TAPI 3.0. It can run remotely and it runs in the client application context rather than in the server context. For setting up media calls, it has a pipe that runs from it, through TAPI, to the TSP, to exchange context about the connections and the application.

Notice that the UI DLL is residing on the client side while the TSP is running in the TAPI server context that is on the server side. This means that any registry writing done by the UI DLL is sent back to the TSP so that the TSP writes to the server, which is the logical place, rather than on some random client that may not be available the next time the application runs.

One issue that must still be considered in TAPI 3.0 is how security should be applied in a distributed environment. The TAPI 3.0 security scheme is line-based rather than address-based as in TAPI 2.*x*. This should be taken into account when modeling both hardware and services, and when deciding how to split them up among addresses, lines, and phone devices. For example, if you decide to have one line with many addresses, security can be applied only to the line and there is no way to distinguish among the addresses. On the other hand, if you decide to have a separate line for each address, then security can be applied in a much more granular fashion.

Another point of similarity between TAPI 3.0 and TAPI 2.*x* is that, on Windows 2000 servers, TAPI3.DLL in TAPI 3.0 is a peer to the TAPI 2.*x* TAPI32.DLL. A TAPI application can load either TAPI32.DLL or TAPI3.DLL. Finally, TAPI 2.*x* TSPs will work with TAPI 3.0. TAPI 3.0 enables a TAPI 3.0 application that is connection-only to transparently pick up the TAPI 2.*x* TSP.

A Review of TAPI 2.*x* SP Objects

The TAPI 2.*x* objects that are of interest to an SP are, hierarchically:

- Line Devices
  - Addresses
    - Calls
- Phone Devices

Line devices own addresses and addresses support calls. On the same level as line devices are phone devices. Several new objects have been introduced in TAPI 3.0. One of these is streams. Streams are operated on when setting up sources and sinks for media. Calls support streams and streams provide access to terminals. Terminals act as the endpoints for calls. Streams and terminals are discuss d later in the paper.

As we said earlier, one difference between TAPI 2.*x* and TAPI 3.0 is that TAPI 2.*x* is line-based while TAPI 3.0 is address-based. (In TAPI 3.0, addresses rather than lines are constantly being acquired and queried for their capabilities.) This doesn't affect TSPs because TAPI 3.0 deals with the differences and line devices can still be accessed from address objects.

Modeling TAPI Lines

When providing connection and media service, the first question is how to model devices. TAPI doesn't mandate any particular scheme. It is possible to have a single address per line or multiple addresses per line. Similarly, it is possible to have a single address type per line or many address types.

For example, a telephone with both an ISDN (Integrated Services Digital Network) line and a POTS (Plain Old Telephone Service) line could be modeled as a single line device with a POTS-type address and an ISDN-type address. It would be up to the application to query those addresses to find out their capabilities. On the other hand, the device could be modeled as having two lines. This might make more sense because applications typically pick up a line and assume that every address on that line is of the same type.

These decisions also have repercussions for security. For example, if a PBX (Public

Branch Exchange) is modeled as a single line, where every station on the line has a separate address, then you can't offer any security. On the other hand, if you use an address-based scheme, with a line for every station, you can grant users access to their individual lines.

Also, TAPI 3.0 has new protocol and address types beyond the implicit PSTN-types available in TAPI 2.*x*, discussed later in the paper. These new types are all per-line, which means that all addresses on a line are assumed to be of the same protocol type.

Finding a Suitable Line

In TAPI 2.*x*, an application picks a SP by doing the following:

1. The application calls *lineInitialize/Ex* and iterates through the line devices.
2. It negotiates the version numbers for all of the lines in the application with *lineNegotiateAPIVersion*.
3. It determines the line capabilities with *lineGetDevCaps*.
4. It determines the address capabilities with *lineGetAddressCaps*.
5. It looks for appropriate media and bearer modes with *lineGetCallInfo*.
6. It may also look for **LINEDEVCAPS.dwPermanentLineID**.

TAPI 3.0 objects, which also support streaming, have similar mechanisms for initialization and for discovering capabilities. One enhancement in TAPI 3.0 concerns the **LINEDEVCAPS.dwPermanentID** attribute. Because this attribute is not necessarily permanent or unique in TAPI 2.*x*, and because in a distributed environment, uniqueness is necessary in both space and time, TAPI 3.0 uses a globally unique identifier (GUID), which is guaranteed to be unique across all services.

Opening a Line

At the SPI (Service Provider Interface) level, a line in TAPI 2.*x* should only be opened once—when an application first shows an interest in it. The **TSPI_lineOpen** method should return reasonably quickly. If the open is time-consuming, consider using a thread. Also, if there is an error, first return, and then send a **LINE_CLOSE**.

Once the line is opened, **TSPI_lineSetDefaultMediaDetection** indicates what types of calls the application is interested in. An application can receive calls of the correct type(s) once, many times, or never. It's important that, if the correct call is never received, new calls (of the incorrect type) are never indicated. This is because TAPI will immediately close the call and invalidate the context. The upshot is that, when users pick up the phone to make a call, the SP disconnects them.

A Review of TAPI Handles

There are three types of TAPI handles:

- The h*xxx* handle. This is the application's handle and is generated by TAPI.
- The ht*xxx* handle. This is generated by TAPI and given to the TSP. It describes

the TAPI context.

- The hdXxx handle. This is generated by the TSP and given to TAPI. It describes the TSP objects and TAPI uses the handle to refer to those objects when talking to you.

The second and third types of handles are of the greatest interest. A series of diagrams, shown below, illustrate how an SP uses the handles while processing a call.

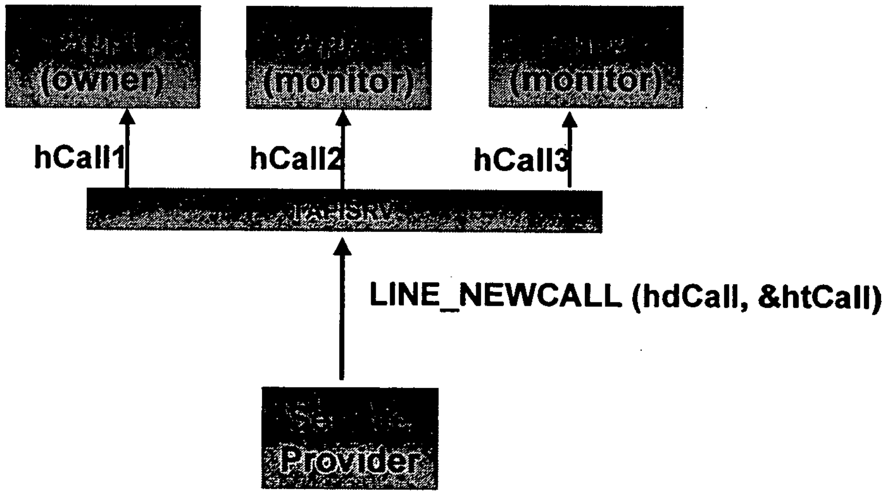


Figure 3. Service Provider processing a call

Figure 3 shows three applications. The first application opens the device as an owner while the other two are only monitoring and won't be expected to pick up any calls and own them.

The **LINE_NEWCALL** method indicates that a call has come in on the device. The method gives TAPI the SP-generated driver handle (the hdCall handle) and also has a place for TAPI to write its own handle to the call (the htCall handle). TAPI then generates call handles hCall1, hCall2, and hCall3 to each of the applications.

The state transition to offering is shown in Figure 4.

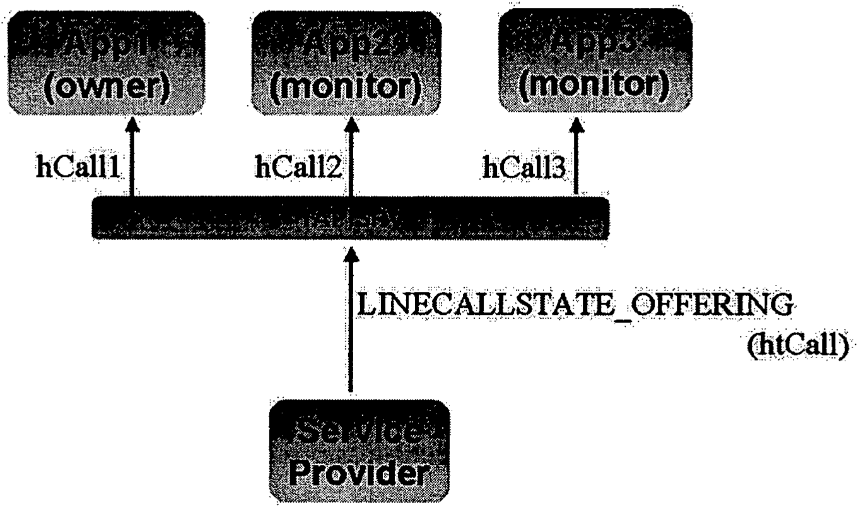


Figure 4. State transition to offering

Notice that the **LINECALLSTATE_OFFERING** uses the TAPI context that it filled in during the previous step. The next diagram, Figure 5, shows what happens when the call is answered.

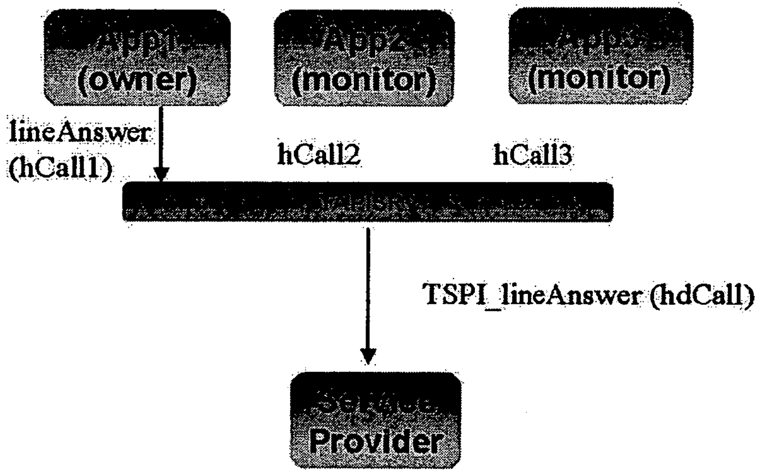


Figure 5. Call answered

The owner application performs a **lineAnswer** and TAPI responds with a **TSPI_lineAnswer**, using the driver call handle to indicate the context. The next diagram, Figure 6, shows what happens when the call state is active.

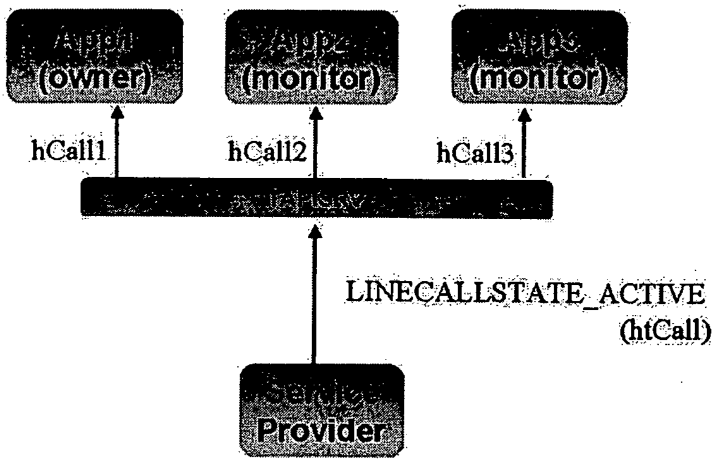


Figure 6. An active call state

Here, the SP responds and changes the state to active. The **LINECALLSTATE_ACTIVE** method uses the TAPI handle (htCall) and all the applications are notified of the state change.

Figure 7 illustrates what happens when a call is dropped.

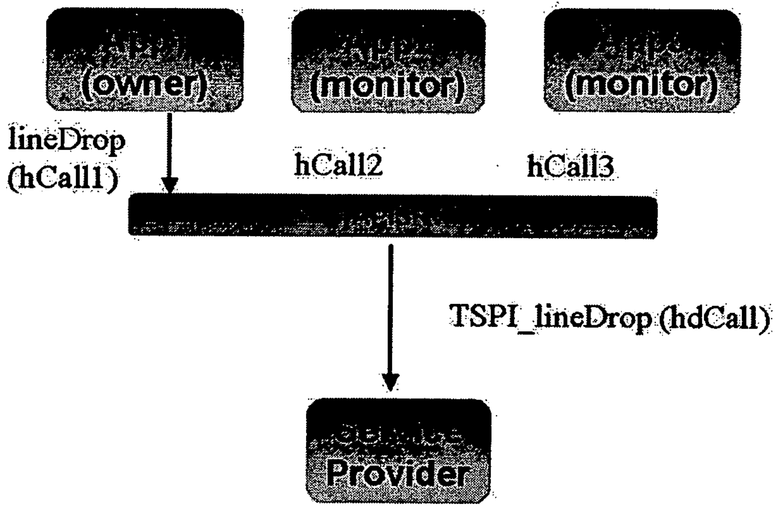


Figure 7. A call is dropped

When the owner application is finished with the call, it calls the **lineDrop** function and the SP receives a **TSPI_lineDrop**, with its own context. The next diagram, Figure 8, shows what happens when the call state is idle.

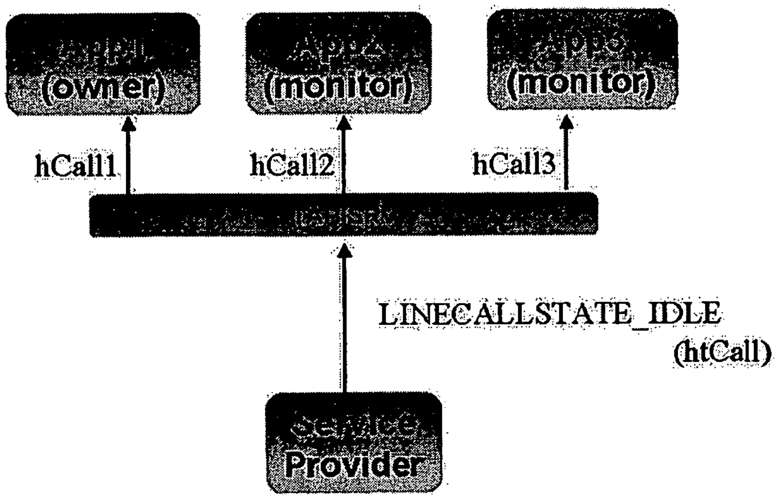
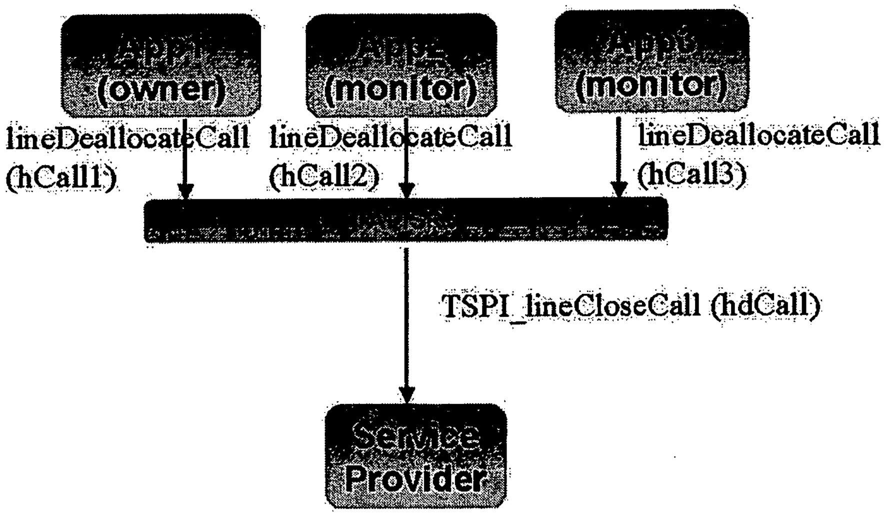


Figure 8. Call state is idle

Once a call is idle, it can't transition to any other state. It is still possible to get **ITCallInfo** information about the call to find out such things as the duration of the call and the parties involved in that call. The call is alive as long as an application has a handle open to it. This means that TAPI isn't going to deallocate the call, the SP call handle, or its own context until the last application has deallocated its application call handle. This is shown in the Figure 9.

When the first application deallocates the call nothing happens, and this is also true when the second application does the same. Only when the third application deallocates the call does the SP receive the **TSPI_lineCloseCall**, again with its own driver handle. Once this has happened, you can no longer rely on any of the contexts surviving. The htCall handle may be reused immediately, which means it is important that such things as tables and pointers no longer reference it.

Figure 9. Deallocating the call


App1 (owner)
App2 (monitor)
App3 (monitor)
lineDeallocateCall (hCall1)
lineDeallocateCall (hCall2)
lineDeallocateCall (hCall3)
TSPI_lineCloseCall (hdCall)
Service Provider

TAPI 3.0 OVERVIEW

Now that you've reviewed the basics of earlier versions of TAPI, you can examine the changes that have occurred in TAPI Service Provider Interface (TSPI) 3.0. It's important to remember that the changes at the TSPI level are incremental. Existing, connection-oriented SPs will work and you can take advantage of some of the new features in TAPI 3.0 without necessarily having to use COM or write an MSP. Examples of this are adding CallHub call control and reporting IP telephony capabilities. If you have a wave device, you can take advantage of the DirectShow integration that is part of TAPI 3.0 by using the MSP that is already included.

The following sections discuss the TSPI enhancements that are in TAPI 3.0. These enhancements include:

- Address types.
- Protocol types.
- A permanent and unique line and phone GUID.
- CallHub support.
- New capabilities reporting.
- MSP support.

Address Types

Address types tell the application what sort of dialable strings the SP supports and also tells the SP the format of those strings. Earlier versions of TAPI always assumed that dialable strings were phone numbers. This is no longer true. There is a session descriptor type for IP telephony and a variety of ways for addressing users, such as by name, machine name, and IP address. Here is the complete list of address types:

- **LINEADDRESSTYPE_PHONENUMBER**, which means the address is a phone number.
- **LINEADDRESSTYPE_SDP**, which is a Session Description Protocol (SDP) address. This protocol is an IETF standard for announcing multicast conferences.
- **LINEADDRESSTYPE_EMAILNAME**, which means the address is an email name.
- **LINEADDRESSTYPE_DOMAINNAME**, which means the address is a domain name.
- **LINEADDRESSTYPE_IPADDRESS**, which means the address is an IP address.

These types are provided to applications in the following structures:

- **LINECALLINFO**
- **LINECALLPARAMS**
- **LINEDEVCAPS**

The **LINECALLINFO** structure indicates the call's address type. The **LINECALLPARAMS** structure tells the SP the format of the destination source strings and the **LINEDEVCAPS** structure holds a variety of information about the line's capabilities such as the different kinds of tones that can be generated or the digit modes.

TAPI 3.0 Prot col Typ s

Just as there are new address types in TAPI 3.0, there are also new protocol types. The protocol types supported in TAPI 3.0 are:

- **TAPIPROTOCOL_PSTN**, which supports voice.
- **TAPIPROTOCOL_H323**, which supports the ITU standard, H.323, for videoconferencing over packet-switched networks such as LANs and the Internet.
- **TAPIPROTOCOL_Multicast**, which supports calls made using the Multicast backBone (MBONE).

Earlier versions of TAPI assumed that call control was always across PSTN but in TAPI 3.0 there is support for IP telephony with the H.323 and multicast protocols. These protocols are GUIDs with one GUID per line. This means that there must be a separate line for each protocol that the SP supports. The permanent line GUID is stored in the **LINEDEVCAPS** structure. There is also a permanent phone GUID stored in the **PHONECAPS** structure.

TAPI 3.0 SPI CallHub Support

Earlier versions of TAPI presented a first-party view of a call, which means the call handle only represented one endpoint of the connection. The notion of a half-call view (in essence, a switch) didn't really exist. In TAPI 3.0, the CallHub object provides the switch's view of a call. This is also called a third-party view. There is one CallHub per conference, with some number of half-calls or call handles connected to it—one for each user. Each of these handles can be manipulated separately.

For current TSPs that use the *dwCallID* field in **LINECALLINFO**, TAPI automatically creates a CallHub for a call. TAPI assumes that every call that references the same ID is part of the same CallHub.

There is also new information available for tracking. The **CALLHUBTRACKING** constants are:

- **LINECALLHUBTRACKING_ALLCALLS**
- **LINECALLHUBTRACKING_PROVIDERLEVEL**
- **LINECALLHUBTRACKING_NONE**

These constants let you monitor CallHubs across all devices on a TSP, across only those devices that have **SetDefaultMediaDetection** selected, or lastly, you can indicate that you don't want TAPI to create CallHubs. One reason to choose this option is because of concerns about security. To retrieve and set the type of CallHub tracking, use the *TSPI_lineGetCallHubTracking* and the *TSPI_lineSetCallHubTracking* functions.

TAPI 3.0 SPI LINEDEVCAPS Flags

The following constants have been added to the **LINEDEVCAPFLAGS** structure:

- **LINEDEVCAPFLAGS_MSP**, which reports if the line has MSP capabilities.
- **LINEDEVCAPFLAGS_CALLHUB**, which reports if the line has CallHub capabilities.
- **LINEDEVCAPFLAGS_CALLHUBTRACKING**, which reports if the line has CallHub tracking capabilities.
- **LINEDEVCAPFLAGS_PRIVATEOBJECTS**, which reports if the line has any private objects.

Private objects allow you to perform tasks that are specific to your SP. Some device-specific operations are a matter of implementing additional private interfaces on the SP. (You're free to expose whatever interfaces you like.) These are cases where the interfaces are exposed directly to the application and are not aggregated. An example is making a terminal object behave any way you like.

In other cases, private objects are actually aggregated by TAPI into TAPI 3.0 standard objects, again allowing you to do things that are specific to your SP. For instance, a Call object could contain your object's private data, allowing you to treat a call in some non-standard way.

TAPI 3.0 SPI MSP Support

Enhancements in TAPI 3.0 for supporting MSPs include:

- A function to find out the MSP CLSID, which is used when creating an MSP.
- A function to open a communication channel between the MSP and TSP.
- A channel function message and response system. This allows an opaqu structure to be passed back and forth between the TSP and the MSP.

The communication channel allows an MSP to set all the characteristics of a call before a connection is made. Typically, the application first sets up the DirectShow filters, the terminals, and any other conditions that it requires. Once this is done, TAPI asks the TSP to actually establish the connection. Initially, then, it is usually the TSP that uses the communication channel to query the MSP to find out the characteristics of the call. The MSP uses the channel to indicate to the TSP when those characteristics have changed.

An Overview of MSPs

An MSP is a COM-based object used to construct application-specific streams that are exposed through terminal objects. An MSP is similar to the UI DLL in TAPI 2.*x*. It is loaded into the application process because it constructs the streams and terminals, which means it must be available to that instance of the application. One way to think of an MSP is as a part of the TSP that operates in the application process. MSPs and TSPs are tied together 1-to-1. For example, there is an H.323 TSP and MSP. These can't be mixed and matched because the communication between the two is opaque. The following diagram, Figure 10, shows the MSP object model.

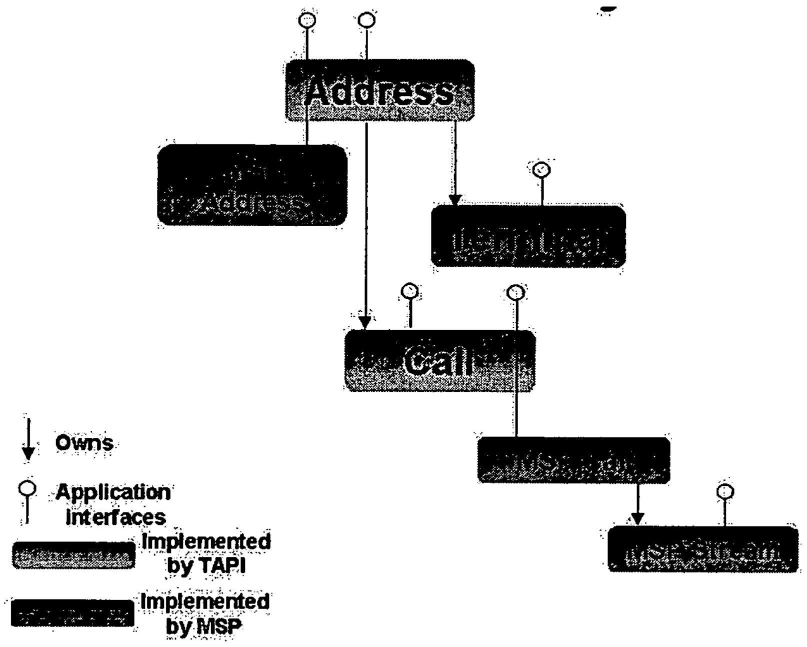


Figure 10. MSP object model

TAPI first creates an address object for each TSP address. This means that for each line device that exposes an address, TAPI creates an object. This object provides an application's access to the various services.

If the TSP has an MSP associated with it, then for each address object, TAPI asks the MSP to create an MSP address object. This is aggregated through the TAPI address object so that its interfaces become available to the application. The MSP address object is used to enumerate the terminals and these are exposed directly to the application by TAPI-defined interfaces.

When a call is created, TAPI first creates its own call object but it also requests the MSP to create an MSP call object, which TAPI aggregates. Finally, the stream objects are created from the MSP call object. Once again, the application can access these interfaces directly.

TAPI 3.0 Interfaces

All the TAPI 3.0 interfaces are illustrated in Figure 11.

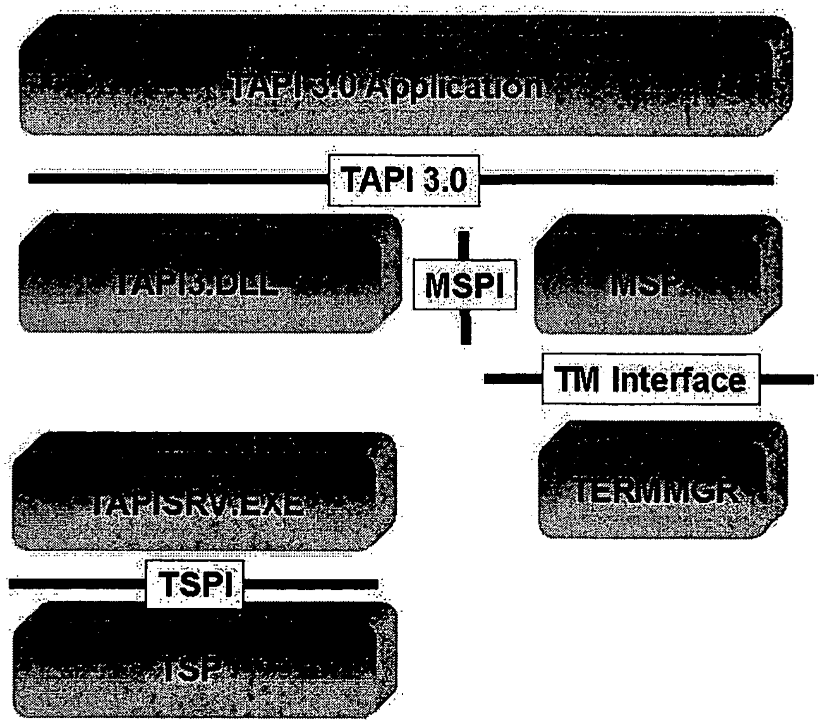


Figure 11. TAPI 3.0 interfaces

Writers of SPs are particularly interested in the TSPI interface to tapisrv.ex , the MSPI interface to TAPI 3.0 and the application process context, and the terminal manager interface by which existing terminals can be discovered, built, and connected to the DirectShow filtergraph.

Note that, in the diagram, the TAPI 3.0 interface extends over the MSP. This is to indicate that the MSP exposes objects that implement TAPI interfaces. It doesn't mean that TAPI is actually aggregating those objects. Because the MSP does implement TAPI interfaces, it's important that those interfaces be implemented in a way that's compatible with HTML-based scripting languages such as Visual Basic® Scripting Language (VBScript).

An Overview of Terminals

Terminals are devices that are the final source or sink of media on a call. They are usually implemented by an MSP and can be divided into two types—static and dynamic. Static terminals are typically resource-intensive and nonshareable. Examples of static terminals include sound cards, microphones and speakers. Dynamic terminals are usually only limited by hardware resources such as memory. Examples of dynamic terminals include video windows and DTMF (Dual Tone Multi-Frequency) detection and generation. The TAPI3.DLL implements terminals corresponding to TAPI phone devices.

An application can select a terminal on a stream at any point in the lifetime of a call. (This means before the SP receives a **TSPI_lineCloseCall**.) Although it's possible for the MSP to fail a request for a terminal, the preferred method is that the MSP be able to select and deselect terminals on the fly.

In TAPI 2.*x*, most SPs had a wave device for accessing the media streams. TAPI provided a generic way for the application to retrieve the device (the **lineGetID** method) but after that it was up to the application to handle media control. However, many applications would prefer to simply direct the bitstream rather than operate on it. To relieve applications from actually having to deal with the bits, TAPI 3.0 has transferred these responsibilities onto the MSP, which, in turn, uses the capabilities of DirectShow for handling the raw bitstream.

Terminal interfaces are designed to define specific tasks that an application may want to perform. They provide a standard model for media control and simplify most streaming tasks such as the following:

- DTMF.
- Speech recognition and text to speech.
- Recording files and playing them back.
- Interactive audio and video.

Acting as COM interfaces, terminals represent a contract between the media streaming implementation and the application. They guarantee that media control in applications will work across various TSPs. The abstract layer remains the same, even if the media streaming tasks are implemented differently under the surface.

An Overview of Streams

A stream represents a single media type and a single direction on a call. (Note that it is not at all uncommon for a single call to have multiple streams.) An application selects terminals on streams. Doing this tells the MSP how to set up its media streaming. The following drawing, Figure 12, illustrates streams on a full-duplex audio/video call.

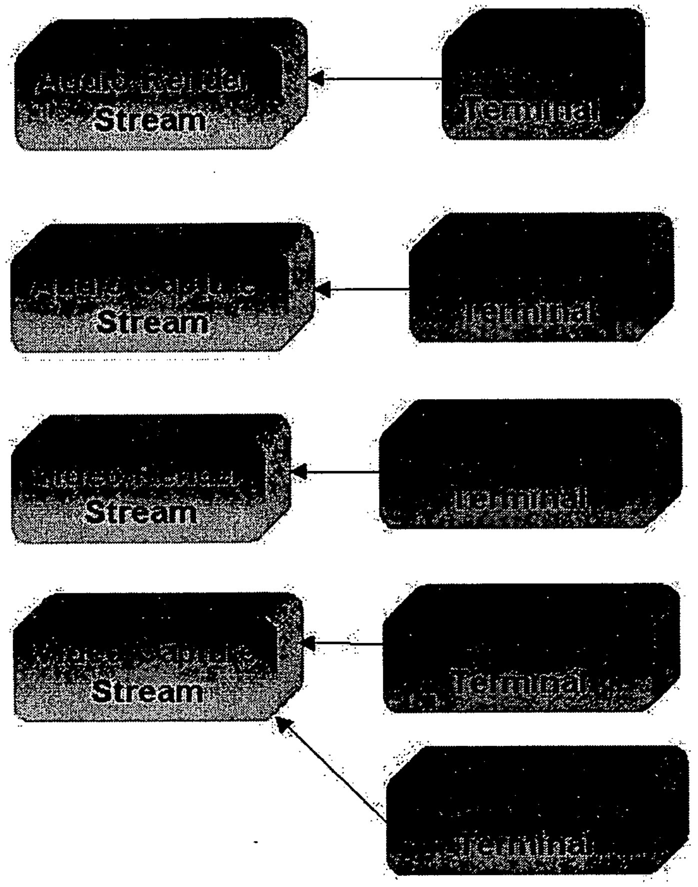


Figure 12. Stream on a full-duplex audio/video call

This call has four streams:

- An audio render stream.
- An audio capture stream.
- A video render stream.
- A video capture stream.

Through these streams, you can plumb in a speaker terminal to render the audio, a microphone terminal as a source for the audio, a video window terminal, a video camera terminal and finally, another video window terminal on the video capture stream. This last window lets you see what is going out over the network and it means you need a capture stream for the render video stream.

Cases where directions are contradictory are not uncommon and the MSP should be able to take a stream that is going in one direction, demultiplex it, and push it in the other direction. Of course, it is possible to simply return an error, but the preferred method is that the MSP be able to handle the situation. Another example of contradictory streams could be rendering audio to multiple locations, such as to a speaker and to a file while recording a conference.

Streams provide applications with a standard model for dealing with media streaming on a call. They allow applications to tell the MSP how to set up media streaming unambiguously. Streams separate the media from the call and divide it into four parts:

- The media type.
- The media direction.
- The media source.
- The media sink.

This means there is a much finer level of manipulation through streams than was possible through the call object. Distinguishable sources and destinations can always benefit by being exposed through separate streams.

Streams support substreams, which are exactly like streams except that they are one level lower. An example of substreams, used by the IP multicast capabilities supported in TAPI 3.0, is dividing an incoming video stream into substreams, where each substream represents one of the participants in the conference. You can then apply a separate video terminal on each substream to display each of them in different windows.

The substream is part of the stream object's capabilities. It is implemented by the MSP and is directly exposed to the application, not aggregated through TAPI. An MSP can implement private interfaces to gain even more control over a substream. For example, in the multicast example, an MSP could implement a participant interface to further identify conference members.

An Overview of the Wave MSP

The WaveMSP is a generic MSP provided by Microsoft that can be used with any TSP that has a wave device. In earlier versions of TAPI, applications needed to use **getLineID** to retrieve the wave device ID. They then needed to open the wave device, which would be locked up for the duration of the session, and they had to handle all the media control themselves. The WaveMSP is an out of the box solution. During initialization, TAPI queries the TSP for a wave device. If one exists, then the WaveMSP is used. The WaveMSP wraps the wave device to fit into the TAPI 3.0 object model. The WaveMSP exposes the DirectShow interfaces directly above the wave device. It uses the Terminal Manager to discover, create, and use terminals.

Determining the Need for Writing MSPs

To summarize, the functions of an MSP are the following:

- It implements terminals.
- It communicates terminal selection with the TSP.
- It performs media control (the TSP handles call control).
- It can set up streaming for client application in the application context

There is no reason to write an MSP if only call control is needed and not media control. There is also no reason to write one if the TSP already has a wave device.

In this case, use the WaveMSP. The only reason not to use the WaveMSP would be to take advantage of some specific feature by writing new interfaces that the WaveMSP doesn't provide. For example, if there is very fine granularity for measuring progress detection on a device, you may want to implement this as a feature on the stream, using your own MSP. Of course, the most general reason to write an MSP is to provide TAPI 3.0 applications with a finer degree of control of media on the telephony device.

The TAPI 3.0 Algorithm for Terminal Creation and Disc v ry

The way TAPI 3.0 discovers and creates terminals is summarized in the following piece of pseudocode:

```
if the TSP has an MSP
        enumerate terminals from MSP
else
        if the TSP has a wave device
                enumerate terminals from WaveMSP
        else
                create terminals based on phone devices
```

TAPI first checks to see if the TSP has an MSP. If it does, then TAPI asks the MSP to enumerate the terminals. If there is no MSP, TAPI then checks to see if the TSP has a wave device. If it does, TAPI uses the WaveMSP that enumerates the terminals, discovering what hardware is available for supporting audio. Lastly, if there is no MSP or wave device, TAPI creates terminals based on the phon devices that are available. TAPI creates terminal and stream objects for phone devices, so that you can direct the media stream to your phone device on your call.

An Overview of MSPI

The Media Stream Provider Interface (MSPI) allows the MSP to enumerate and create terminals for the application. The MSPI is used to create the MSP Call object, which is then aggregated. Once this happens, the application talks directly to the MSP, bypassing TAPI. It selects terminals the MSP has created, on streams that the MSP has implemented and created. Because the application talks directly to the MSP, it's important that the MSP conform to the defined TAPI interfaces and be scripting compatible.

An Overview of the Terminal Manager

The terminal manager, like the WaveMSP, is intended to simplify MSP development. The terminal manager makes it easier to write MSPs, to interface with DirectShow, and to locate hardware resources on the computer.

The terminal manager is a small helper DLL that enumerates the static terminals and also has base classes that you can derive from when writing MSPs. These classes are designed to make it easy to write MSPs based on DirectShow. They do the following:

- Enumerate and create DirectShow terminals.

- Insert filters into a DirectShow filtergraph.

The base classes create terminal objects that correspond to found devices that are of concern to TAPI. The helper functions insert filters, based on these terminals, into a DirectShow filtergraph.

Although the terminal manager is based on standard DirectShow devices, you can use whatever you like, creating your own terminal objects and implementing your own interfaces. You can also define additional interfaces on standard DirectShow terminals that an application can query for and use.

Making a TAPI 3.0 Call

This section shows how to establish a TAPI call from the SP point of view. Remember that a call can be voice along with other media streams. The steps are as follows:

1. The application selects an address offered by the SP.
2. The application asks that the SP enumerate the terminals on that address.
   a. TAPI passes this request on to the MSP.
3. The application creates a call on the address: Creating a call means that it is creating a Call object, not actually establishing a connection. To create a call object:
   a. TAPI informs the MSP of the call.
   b. The MSP creates the MSP Call object.
   c. TAPI aggregates the MSP Call object into the TAPI Call object.
4. The application performs a **QueryInterface** to acquire the stream control interface.
   a. Through aggregation, this interface is passed on to the MSP.
5. The streams available on that MSP call are enumerated. This involves enumerating objects that are created by the MSP using interfaces implemented by the MSP.
6. A terminal is selected on the stream.
7. Finally, the call is connected. To accomplish this:
   a. TAPI calls TSPI_lineMakeCall
   b. The TSP and the MSP communicate about the call. The TSP uses the pipe through TAPI to track connection states that will affect characteristics of the stream.

An Overview of the ACD Proxy Functions

This section discusses the support at the service level for the TAPI 3.0 Automatic Call Distribution (ACD) proxy functions. (For a more complete description of ACD, refer to the TAPI 3.0 documentation.) These functions offer an interface for call center functions provided by proxy applications running on the TAPI server and provide flexible options for modeling ACD components.

An ACD component can be modeled in these ways:

- Modeled 1-for-1.
- Modeled entirely in software and built against a known PBX or switch. In this case, all the queuing and vectoring is done in the software.
- Modeled as a hybrid. In this case, you can add specific software functionality to an existing ACD component.

There is transparent support for existing TSPs. For example, it is possible to build a proxy application that runs against an existing PBX TSP.

ACD Proxy Components

The following diagram, Figure 13, illustrates the ACD proxy components.

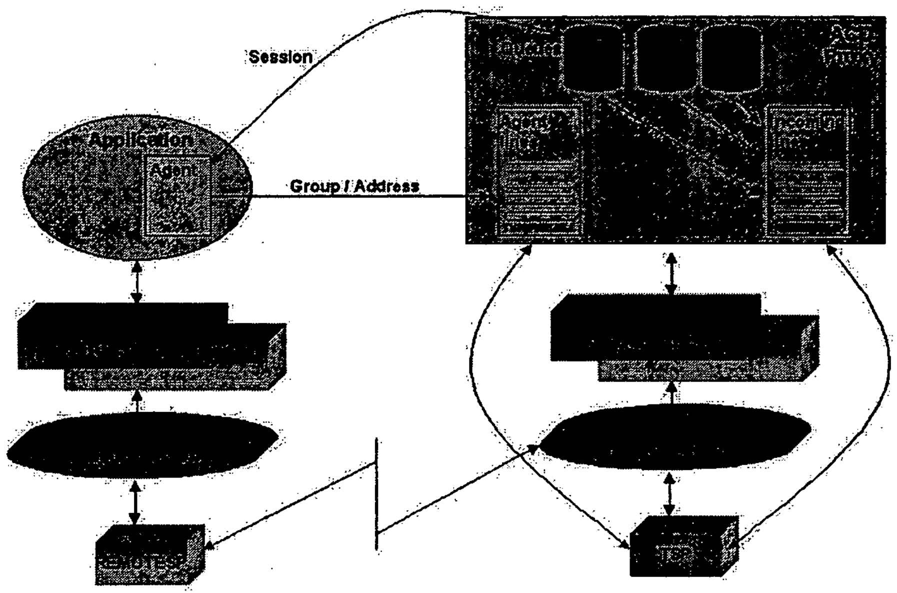


Figure 13. ADC proxy components

The ACD proxy application sits on the server and runs against a regular TSP. The TSP has no knowledge about the ACD system. It only understands lines and addresses.

The application sits on the client. It expects to query for an *ITTAPICallCenter* interface, pick up an agent handler application, register an agent, and finally, to end up in a session, with an address, ready to receive calls.

First the ACD proxy selects lines from the TSP that will be used for incoming calls. These lines aren't seen by the application. They are used only by the ACD proxy for accepting calls. The proxy selects other lines from the TSP that will be used as agent lines. These are addresses that will underlie the agent session. Every session is dynamically assigned a session containing a queue and a group, and an address within that group to which calls will come.

The application also constructs queues. Agents will handle some of these and others will be vectors that are transient and used for such things as announcements. Finally, the session is established and the agent is set up and associated with a particular group and a particular address within that group.

The application communicates with TAPI by function calls that are part of the COM interface in TAPI 3.0. The ACD proxy communicates with TAPI by registering with TAPI through ACD functions. When an application makes a function call, TAPI sends proxy request messages to the ACD proxy. These contain all the arguments that were passed in the application's function call. The proxy responds with response messages that are taken back to the client as function completions. The proxy can also send unsolicited messages for state changes and events.

An Example of Inbound Call Routing

This section gives an example of inbound call routing, using an ACD proxy. Initially, the ACD proxy opens lines for incoming calls, opens lines for agents, and sets **LINEPROXYREQUEST** constants on those addresses. The agent lines are associated with particular agent instances. The ACD proxy receives proxy requests on these lines and responds to those requests on these lines.

To query for an ITCallCenter interface, the ACD client queries with:

```
pTapi->QueryInterface (IID_ITTAPICallCenter, &pCallCenter);
```

The ACD client then enumerates agent handlers with:

```
pCallCenter->EnumerateAgentHandlers (&pEnum);
```

It creates an agent with:

```
pAgentHandler->CreateAgent (&pAgent);
```

It puts the agent in an agent session with a specified group and address, using this call:

```
pAgent->CreateSession (pACDGroup, pAddress, &pAgentSession);
```

Finally, the ACD proxy receives an incoming call on one of its incoming call lines. It selects a queue for the call, giving it the call treatment that the call requires. The proxy next selects a free agent from the appropriate group, probably basing its selection on **ITCallInfo** requests. It either transfers or redirects the call to an agent, depending on whether or not the call was picked up. The client then receives the call notification on an associated address.

CONCLUSION

TAPI 3.0 supports existing TSPs while making it easy to extend their capabilities to support IP telephony. Developers can take advantage of the SPI extensions for adding media control. The WaveMSP provides an out of the box solution to media control for TSPs that already have a wave device. The terminal manager simplifies writing MSPs, interfacing with DirectShow, and locating hardware resources on the computer. By taking advantage of the new ACD functions, developers can provide greatly enhanced call center services.

MORE INFORMATION

For more information about TAPI 3.0 and Microsoft Windows 2000 servers, consult the following Web sites:

For information about TAPI 3.0, go to the Communication Services page of the Windows NT Server Web site.

For information about the family of Windows 2000 servers, go to the Windows 2000 Server Web sites.

You can download the Microsoft Windows Platform SDK.

For sample code, articles, and information about the platform SDK, go to the Microsoft MSDN Web site.

Microsoft

Microsoft Windows 2000 Server

Operating System

Microsoft Web Telephony Engine

Technical Paper

Abstract

The Microsoft® Web Telephony Engine is an open environment that enables Internet technologies and standards to be used to create and execute telephony applications. It enables telephony solutions for both dual-access (browser and telephony) and single-access (telephony-only) Web sites.

The Web Telephony Engine is a run-time engine that uses Hypertext Markup Language (HTML) to enable the use of standard Web authoring tools to create a variety of telephony solutions, such as Interactive Voice Response (IVR), voice mail, automatic call distribution, and call centers.

This technology is consistent with other trends toward open platform use and the convergence of data networking. Web Telephony Engine is a new addition to the Microsoft telephony platform that is included in the Microsoft Platform Software Development Kit for Windows 2000. It complements the Windows Telephony Application Programming Interface (TAPI) 3.0, Windows NetMeeting®, the Microsoft Phone Dialer, and other networking enhancements available in the Windows 2000 operating system. This paper describes the Web Telephony Engine, its applications, and benefits.

C NTENTS

INTRODUCTION

Telephone systems have been traditionally built using proprietary hardware and software architectures. These proprietary telephone systems, and the voice applications designed to run on them, work within a limited capacity, but their closed nature often makes integration between applications expensive and time-consuming, particularly when the applications are made by different companies.

The emergence of the Internet and the widespread adoption of the Internet Protocol (IP) have contributed to the rapid convergence of voice and data networking. Open hardware and software architectures can deliver the reliability and performance needed for mission-critical communications, while providing new levels of application richness.

The open platform model presents significant advantages over legacy, proprietary telephony systems. Closed systems typically require special administrative training and programming expertise, while open architectures embrace a more IT-centric administrative and programming environment. The use of open platforms and industry standards for telephony provides ease of use and flexibility, as well as lower costs and broader choice. This open environment also provides new opportunities for innovation and commercial success for vendors.

Microsoft® is a proponent of the open approach to building telephony systems and solutions. The Windows® operating system family is an important software platform for traditional and IP telephony in client computers, servers, and embedded systems. Windows Telephony Applications Programming Interface (TAPI) allows software applications and telephony hardware made by different companies to work together easily and economically. The Windows operating systems also include Microsoft NetMeeting®, a standards-based IP telephony conferencing and collaboration product. NetMeeting offers a Software Development Kit (SDK) that allows vendors to integrate or incorporate NetMeeting conferencing software with their products.

The Microsoft Web Telephony Engine is an important new addition to the Windows 2000 Microsoft Platform SDK. The Web Telephony Engine serves as an open environment for the development of telephony applications using standard Web resources. The Web Telephony Engine promotes the convergence of IP and traditional telephone networks by enabling organizations to offer information and deliver services, either over the Internet or by telephone.

WEB TELEPH NY EN INE DESCRIPTION

The Web Telephony Engine expands the value of existing Web standards and tools to include voice application support. This technology allows developers to use HTML, Active Server Pages (ASP), and other Web authoring tools to develop telephony applications.

The Web Telephony Engine makes it easy to create dual-access Web sites—sites that can be browsed using a Web browser application, or by the telephone, using speech or dial-pad navigation commands.

The Web Telephony Engine runs on Web servers and integrates with TAPI 3.0 and the Microsoft Speech API (SAPI).

The Web Telephony Engine is based on five simple extensions to HTML that enable rich voice processing capabilities. Microsoft has submitted these extensions, as well as the Web Telephony Engine itself, to the World Wide Web Consortium (W3C) Voice Browser Group for consideration as an industry standard.

Other approaches to using Web technologies for telephony application development have been proposed including an XML schema called VXML (or Voice XML). Microsoft continues to work with the vendor community and standards organizations on the broad use of XML schemas to enable application integration in a number of areas. However, the Web Telephony Engine offers some specific advantages over the proposed VXML schema.

The Web Telephony Engine provides an easy technology path by enabling HTML to transparently and simultaneously support telephony services in conjunction with other Web site services or standalone voice applications. This approach eliminates the need to invent a new standard or language to enable voice-based services on the Web.

The use of standard Web technologies lets standard Web servers host or render voice applications. This flexibility fosters a more open market for telephony applications than has previously existed, and creates new opportunities for system integrators to provide telephony functions to their customers.

More than 90 developers have worked with Microsoft through the developer beta process leading to the completion of Web Telephony Engine. These vendors are at various stages of readiness with their own software and hardware products using the Web Telephony Engine. Microsoft plans to continue working with these vendors and the broader industry through the W3C.

The Microsoft Web Telephony Engine includes the following components:

- A run-time engine that delivers HTML-based processes over the telephone.
- A management interface that controls the run-time engine.
- Full Telephony User Interface (TUI) implementation of pertinent HTML tags.
- TAPI 3.0 scripting integration that enables telephony operations on a Windows 2000 Server-based network.

TELEPH NY APPLICATIONS

TAPI 3.0 enables developers to use a variety of programming languages to create their telephony applications. The Web Telephony Engine supports this language-neutral approach for a variety of applications.

Language-Neutral Support

Because TAPI 3.0 is a Component Object Model (COM)-based API, it can support a variety of programming languages—not just C++. As a result, developers can use the programming language of their choice, such as Visual C++® development system, Visual Basic® Scripting Language, or JScript® development software. This flexibility increases the number of developers who can telephony-enable their applications.

The Web Telephony Engine provides the environment for using HTML to develop telephony applications, making the development and deployment of the telephony solution faster, simpler, and less expensive.

Because it is based upon existing Web technologies, the Web Telephony Engine can enable interaction easily with an Internet site or application over the telephone. Dual-access Web sites extends the usefulness of an organization's Web presence.

The following section explains how telephony applications work, how they relate to Internet applications, and how the Web Telephony Engine can be used towards development and deployment of these applications.

Interactive Voice Response

A common and representative telephony application is Interactive Voice Response (IVR). An IVR application is a set of rules and commands that:

- Leads a telephone caller through a system of menus, with navigation enabled by telephone dial pad or speech commands.
- Delivers information to the caller in the form of a voice response, based on the choices the caller makes.
- Collects voice and data inputs, and often allows a user to interact with and make changes to a live database.
- Performs other defined operations on behalf of the caller or the program sponsor.

IVR technology allows organizations to provide cost-effective, around-the-clock, self-service support by telephone for their customers, partners, or employees. For example, IVR enables financial institutions to allow telephone banking, so customers can determine the status of deposits, withdrawals, and check clearance, as well as transfer funds, pay bills, and so on.

IVR-like functions can also be used in auto-attendant applications. An auto-attendant is a software application that prompts a caller for the name or extension of the party the caller wishes to reach within an organization. The caller can respond to these prompts by speaking or by using the telephone dial pad, depending on the auto-attendant application's capabilities.

In addition, IVR systems are often used as a front-end to an inbound call center. The IVR system intercepts the incoming call and provides the caller access to desired information, or conducts the desired transaction in a self-service manner. If the caller still needs to talk with a live operator, the IVR system can route the call to another part of the phone system (usually an automatic call distributor), which in turn routes the caller to the next available or most appropriate agent. IVR systems can substantially reduce the number of calls that live operators must handle, thus reducing an organization's costs while providing high quality customer service.

Web Site Similarities to IVR

An IVR application is similar to an Internet Web site in which HTML links are used to navigate through the site; forms are used to collect information, and ActiveX® controls, or Java applets are used to perform more complex operations. Because of these similarities, the Web site is a suitable model for IVR applications and the Microsoft Web Telephony Engine.

Businesses, particularly small ones, are faced with a set of inter-related and often conflicting communication challenges. They need to find ways to:

- Control customer service and related telephone support costs while improving customer satisfaction.
- Improve communications between internal departments and customers without investing in a complex, difficult to maintain, and inflexible voice mail system.
- Compete with larger competitors who can justify a larger investment in IVR.
- Be confident that the message delivered on their IVR system is consistent with the message published on their Internet site.

IVR applications and Web sites are two methods by which organizations can address these issues. The Web Telephony Engine provides a new way to address these application areas using common technologies.

Using Web Telephony Engine

Using the Web Telephony Engine, organizations can develop Web sites that provide product information and order forms. Web sites today can be browsed using standard Web browsers. The Web Telephony Engine enables a Web site to be additionally accessible by the telephone exclusively, or, in the case of a telephony application, by a dual-access Web site.

Access to these sites can be facilitated by telephone, using voice commands or pushing telephone dial-pad buttons. Speech technologies can process speech commands to select options from menus. Speech recognition technology enables spoken words to be converted into text or numbers, which can then be entered as fields in a transaction or database. Text-to-speech technologies enable the content of a database or Web site to be read to the caller over the phone. These technologies can facilitate enhanced customer support.

Users hearing a voice prompt over a telephone and interacting with that service using speech or dial pad commands do not know or care that the voice prompt and other interaction is initiated by HTML code running on a Web server. The user is simply accessing the information or completing the desired transaction over their telephone.

Organizations are always looking for new ways to provide better customer service. Although organizations continue to provide customer service over the telephone, the Web has emerged as a strategic and important vehicle for providing additional means of customer support. Because far more people have access to a telephone than to a computer, the Web Telephony Engine allows the Internet or an intranet site to be experienced by many people who might not otherwise receive its benefits

As organizations seek new and better ways of providing customer service, the focal point of this customer service—the call center—is changing. Historically, call centers staffed by customer service agents have handled incoming and outgoing telephone calls. Now these agents must handle customer inquiries from increasingly varied sources.

Customer service agents who previously answered phone calls and fax inquiries must now answer e-mail and perhaps participate in Web chat rooms or instant messaging sessions with customers. Voice-over-IP interaction is being used increasingly on the Web for customer interaction and collaboration.

BENEFITS

The Microsoft Web Telephony Engine provides a number of benefits for developers, users, organizations, and vendors. It enables the generation of a variety of new applications and provides enhancements to existing applications. The Web Telephony Engine provides profitable opportunities for organizations, giving new and small developers and vendors a chance to compete in the marketplace created by the convergence of voice and data. All of these elements combine to create a greatly enhanced user experience.

Supports Dual and Single Access Sites

Corporations maintain both Web sites and telephony systems for communicating with their customers. Each system requires the maintenance of a separate staff and set of platforms. The integration provided by the Web Telephony Engine lets corporations maintain a single customer service application, reducing development and maintenance costs.

When used in the Web Telephony Engine, the same HTML code can simultaneously render a Web site through a browser or a telephone set. A single Web site can be made accessible by graphical user interface and by voice. The code can be additionally combined with Web authoring paradigms such as HTML, VBScript, JScripts, ActiveX controls, Java Beans, and Active Server Pages and CGI in the creation of telephone-accessible-only Web sites.

The Web Telephony Engine allows flexibility relative to which parts of any page or Web site is accessed by what method. While some applications such as voice mail and call routing are telephony oriented, other applications such as news services, banking, or brokerage systems are more suited for Web browsers. The Web Telephony Engine lets organizations make their data available over any preferred media. Web sites can be made accessible using the telephone or industry standard browsers. Because IVR and browser navigation are often very different, special single-mode sites can be created that are optimized for use by one method only.

Universal Access

An important aspect of telephony development is the expansion of Web access to people with disabilities. The United States Rehabilitation Act Section 508 requires that federal agencies purchase electronic and information technology (EIT) that is accessible to people with disabilities. These standards apply to a full range of technologies used for communication. Such technologies must be accessible and usable by people with a wide range of disabilities, including: visual disabilities, such as blindness, low vision or lack of color perception; hearing disabilities; speech disabilities; or physical disabilities, such as limited strength, reach, or dexterity.

The Web Telephony Engine assists people accessing a Web site or a Web-based service by providing access using a telephone or other telecommunications device, using either speech or telephone dial pad commands.

Distribut d Syst ms Ec n mics

Over the years, the computer industry has migrated from mainframes and minicomputers to powerful standalone desktop computers and distributed client-server computing. Now, client-server architectures are giving way to *n-tier* distributing computing models. N-tier computing is a term associated with the next logical step in distributed computing beyond traditional client-server architectures. N-tier implies that a Web layer might be used to enable a client user experience. This Web layer serves as a front-end to several back-end processes that could be running on a variety of machines in different locations.

Web Telephony Engine can enable n-tier voice and Web site deployments, moving voice processing or rendering services to the most optimal point in a network to reduce bandwidth requirements and reduce costs.

The ability to distribute functionality as needed provides performance, scalability, and manageability benefits. This model allows scalable distributed applications to be created, including large-scale commercial applications such as powerful call center applications that work across multiple sites.

This distributed computing model lets most program functionality exist on the Web server or on back-end servers, rather than on the client. Any number of clients can be connected to a single Web site and receive the same data and interface. Upgrades take place on the server only. Functionality is available to those clients who can benefit from it, without impairing the operation of those that do not need that functionality. The use of an intelligent HTML tag enables a user experience that is appropriate to the device used to access the content. For example, a standard Web browser ignores the voice tags that are appropriate for rendering content for a telephone.

Benefits and Opportunities for Developers

The Microsoft Web Telephony Engine uses both dynamic and static HTML for Web-based telephony application development, and can additionally benefit from the use of Visual Basic, Java, ActiveX, ASP, and other programming tools and languages. Designed to facilitate links between documents, HTML is an effective method of creating the nested menus often used by telephony applications.

Using the Web Telephony Engine with HTML simplifies design, development, and deployment processes by providing a familiar, flexible, and widely used programming standard. This allows developers to efficiently design, publish, and refine telephony applications.

The Web Telephony Engine insulates the application developer from the details of computer-telephony and Web site integration. As a result, creating voice applications is faster and easier to do—it is simply a matter of creating an HTML-based Web site.

Because HTML is widely available, widely used, and well understood, organizations can extend the capabilities of in-house staff to include telephony development with little or no re-training. Rather than having to find a specialized programmer who works with C or other programming language, an organization can locate a wider number of people who can work with HTML to telephony-enable existing applications or to create new ones. A small developer, value-added reseller (VAR), or system integrator who knows HTML can customize telephony applications for vertical or horizontal markets, such as insurance companies, small businesses, and the like. With only slight modifications, one application can serve many types of customers.

Many small businesses or departments within a larger company have not yet enjoyed the benefits of IVR. IVR applications can offer an automated solution for handling many calls if PC-based solutions are readily available and can be easily adapted to fit changing needs.

The Web Telephony Engine creates a framework for traditional telephony vendors to further integrate the Web into their solution sets. Because computer-telephony and Web site integration are extracted, telephony vendors can focus more strictly on development, debugging, analysis, and deployment tools.

The Web Telephony Engine eases the task of developing telephony-enabled Web applications, lowers development costs, and speeds time to market. These advantages allow the creation of new and innovative products that were not feasible using proprietary development methods or architectures.

Developing Telephony Applications on the Windows 2000 Server Platform

HTML applications created with the Microsoft Web Telephony Engine can run on most server platforms, but the development process is accelerated, simplified, and enhanced by the features available in the Windows 2000 Server operating system. This integration with Windows 2000 services further extends the value of the Web Telephony Engine.

Windows 2000 Server includes a number of services and technologies that make it ideal for running communications solutions:

- **Active Directory™** directory service enables policy-based administration of devices and resources across a network.
- **Quality of service (QoS)** support aids in prioritizing the flow of traffic across a network. This quality of service support is provided across routed and LAN switched networks and across the public Internet using the appropriate standards for each environment (such as Resource Reservation Protocol (RSVP), 802.1p, and Differentiated Services).
- **Native Asynchronous Transport Mode (ATM) support** is a standard feature in the Windows 2000 operating system.
- **Performance improvements** enable faster networking and applications processing.
- **Better reliability and scalability** make the Windows 2000 operating system an

even more trusted environment for mission-critical applications.

- **Complete networking services**, including built-in remote access service, LAN and WAN routing, Virtual Private Networking, DHCP service, better enable quick and secure access to information at any time from anywhere.
- **Integrated application environment** with Windows 2000 Distributed interNet Architecture (Windows DNA)—a comprehensive, integrated platform for developing and operating state-of-the-art distributed Web applications.

Running on the Windows 2000 Server platform, the Web Telephony Engine takes advantage of these and other operating system features and services. The Web Telephony Engine can access the Internet Information Server (IIS) using the HTTP protocol to render the HTML scripts over the telephone. With just a few conditional tags, data on an intranet or Internet site can be made available over the telephone, even if the interaction with the Web server is made through a QoS-enabled ATM or IP-based LAN or WAN. Even in cases where different data navigation paths are used for Web and telephone access, large blocks of data can be easily accessed from a single site.

The Web-authoring paradigm is the preferred method of integration for all Microsoft BackOffice® applications. By combining TAPI devices and SAPI-based speech engines on a Windows 2000 Server-based network, the Web Telephony Engine facilitates a single point of integration for these two important standards in the telephony industry. The availability of a solution based on TAPI and SAPI encourages the creation of products based on these standards, expanding the market for these products, and resulting in lower prices and higher availability for the user.

TAPI Integration

TAPI is the API within the Windows operating system family that allows telephony-enabled software written by one vendor to work with telephony hardware made by another vendor. TAPI is supported in the Windows CE, Windows 95, Windows 98, Windows NT® Workstation 4.0, Windows NT Server 4.0, Windows 2000 Professional, and the Windows 2000 Server operating system family.

The Web Telephony Engine uses TAPI to integrate with the telephone system. TAPI 3.0 with COM lets developers choose from a variety of programming languages to develop telephony solutions; and the Web Telephony Engine technology extends this to include development in HTML. The Web Telephony Engine lets HTML work through TAPI to communicate with the underlying network architecture. This resembles the way in which HTML works with the Windows Graphical Device Interface (GDI) to provide a standard display model for users.

This integration provides ease and speed of development while benefiting from TAPI features, including the voice-over-IP support included in TAPI 3.0. Many of the technologies used in Web development, such as security and authentication, database access, and unified messaging, apply to telephony as well. Since many vendors provide ActiveX or Active Server scripting interfaces, their integration with

HTML and VBScript is routine. A special interface transfers the TAPI call information to the ActiveX components that interact with callers or otherwise affect the call progress.

ActiveX Support

Running the Microsoft Web Telephony Engine on a Windows 2000 Server-based network allows companies to create ActiveX controls, which can then be easily integrated with any development tool built for Web-based telephony.

ActiveX components can be integrated with scripts. For example, telephone-specific ActiveX components running in the context of the run-time engine can provide object-oriented access to telephony equipment. Third-party vendors can publish their code as ActiveX components, and receive call context information from the run-time engine. The functionality of ActiveX components is accessible using Visual Basic Scripting Language and JScript.

Support for Vendors

The Web Telephony Engine encourages Independent Software Vendors (ISVs) and Independent Hardware Vendors (IHVs) to work cooperatively to ship better tools and customer solutions. As standardization of content, speech engines, hardware, telephony technology, and development tools evolves, development opportunities correspondingly increase.

The Web Telephony Engine offers a platform to facilitate cooperation between diverse industry sectors, opening opportunities for the development of:

- TAPI drivers for voice boards, ISDN BRI, ISDN PRI, station emulation boards, and switches (traditional or PC-based).
- SAPI interfaces for speech boards or speech engines.
- Development tools to facilitate the creation of dual-access Web site or Web-based telephony applications including monitoring and analysis tools.
- Telephony technologies (ActiveX components).

The Web Telephony Engine helps move the telephony hardware market towards the use of less expensive, industry-standard platforms and hardware, such as network interface cards, modems, and graphical display boards. Telephony card makers today already offer TAPI service providers for their telephony cards. A TAPI service provider is driver software that enables a telephony card or a phone system to understand and respond to TAPI-driven commands from applications. Telephony cards that support TAPI can support any Web Telephony Engine application. In the same way, the Microsoft Speech API (SAPI) provides a standardized way to access speech engines, facilitating the development of powerful speech-enabled applications and services.

Microsoft is currently working with a number of communications card vendors to help them support the Web Telephony Engine. The recent addition of TAPI service providers to the Certified for Windows Logo Program is contributing to this effort.

because the logo program provides a standard benchmark for performance and interoperability for these TAPI service providers. This program should provide a more consistent environment for running TAPI-based applications, and make the installation and ongoing operation of applications with these TAPI-based cards easier.

Because the Microsoft Web Telephony Engine is open and standards-based, vendors need only comply with the published set of HTML attributes and the telephony-specific ActiveX interfaces available in the Windows 2000 Server operating system and the Microsoft Platform SDK. This system can benefit:

- Authorized resellers, who tailor custom solutions.
- Vendors of Internet management tools, who develop capacity planning, statistical analysis, and other management tools.
- Software vendors, who provide telephony application generators, or Internet authoring tools.

Vendor-supplied components can be used together with HTML tools to create dual-access Web sites or pure telephony applications, including:

- IVR
- Auto-attendant
- Audiotex
- Call center solutions
- Messaging solutions
- A variety of vertical markets

SUMMARY

The current migration toward open standards, and the convergence of voice and data creates many new opportunities for vendors, developers, and organizations. Microsoft has developed the Web Telephony Engine to provide the convenience and familiarity of HTML authoring to telephony development. By creating an environment in which HTML scripting can be used in conjunction with VBScript, JScripts, ActiveX components, and other methods, the Web Telephony Engine opens and expands the telephony development arena and marketplace.

Although Web sites enhanced with the Web Telephony Engine can run on essentially any server, development is enhanced by tight integration with the features and applications of the Windows 2000 Server operating system. This includes a graphical administrative interface that facilitates the linking of Web-based telephony sites to TAPI-enabled hardware.

HTML authoring provides benefits to telephony development in the following ways:

- Availability of many skilled professionals.
- A growing variety of authoring, debugging, and management tools.
- Built-in scalability provided by Internet server technologies.
- A growing number of technologies in the form of ActiveX components and other Internet-related interfaces.
- Acceptance within the ISV and IHV community.

Increasingly, the separate worlds of telephony, computing, commerce, customer service are unifying. The Microsoft Web Telephony Engine is another platform innovation that contributes to this convergence.

AVAILABILITY

The Web Telephony Engine is available now as part of the Microsoft Platform Software Development Kit, released in conjunction with the Windows 2000 operating system. Including the service as part of the Platform SDK allows broad customer and partner access to this important new technology.

The Web Telephony Engine was first included in Windows 2000 Release Candidate 2 (RC2). The Platform SDK release includes updated documentation to reflect the addition of the Web Telephony Engine.

The Web Telephony Engine will be shipped as a royalty-free re-distributable executable in the Platform SDK, and will be covered by the standard Platform SDK license agreement. This means that developers can package the Web Telephony Engine code with solutions that are based on the Web Telephony Engine.

FOR MORE INFORMATI N

For the latest information on the Microsoft Windows operating system family, including Windows 2000 Server, please see the Microsoft Windows Web site.

For information about Windows-based communications, please see the Windows 2000 Communications and Networking sites.

For more information about the Microsoft Platform SDK, please see the Microsoft Developer Network Web site.

For information about the World Wide Web Consortium's Voice Browser working group activities, please see the WC3 Web site.